PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L

A2

(11) International Publication Number: WO 99/14881

(43) International Publication Date: 25 March 1999 (25.03.99)

(21) International Application Number: PCT/US98/19316

(22) International Filing Date: 16 September 1998 (16.09.98)

(30) Priority Data:

60/059,082	16 September 1997 (16.09.97)	US
60/059,839	16 September 1997 (16.09.97)	US
60/059,840	16 September 1997 (16.09.97)	US
60/059,841	16 September 1997 (16.09.97)	US
60/059,842	16 September 1997 (16.09.97)	US
60/059,843	16 September 1997 (16.09.97)	US
60/059,844	16 September 1997 (16.09.97)	US
60/059,845	16 September 1997 (16.09.97)	US
60/059,846	16 September 1997 (16.09.97)	US
60/059,847	16 September 1997 (16.09.97)	US

(71) Applicant *(for all designated States except US)*: INFORMATION RESOURCE ENGINEERING, INC. [US/US]; 8029 Corporate Drive, Baltimore, MD 21236 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: KAPLAN, Michael, M. [US/US]; 4 Ocean Drive, Rockport, MA 01966 (US). DOUD, Robert, Walker [US/US]; 4 Redcoat Road, Bedford, MA 01730 (US). KAVSAN, Bronislav [US/US]; 150 Rosemont Drive, North Andover, MA 01845 (US). OBER, Timothy [US/US]; 9 Birch Lane, Atkinson, NH 03811 (US). REED, Peter [US/US]; 1 Bancroft Avenue, Beverly, MA 01915 (US).

(74) Agent: BODNER, Gerald, T.; Hoffmann & Baron, LLP, 350 Jericho Turnpike, Jericho, NY 11753 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published

Without international search report and to be republished upon receipt of that report.

(54) Title: CRYPTOGRAPHIC CO-PROCESSOR

(57) Abstract

A secure communication platform on an integrated circuit is a highly integrated security processor which incorporates a general purpose digital signal processor (DSP), along with a number of high performance cryptographic function elements, as well as a PCI and PCMCIA interface. The secure communications platform is integrated with an off-the-shelf DSP so that a vendor who is interested in digital signal processing could also receive built-in security functions which cooperate with the DSP. The integrated circuit includes a callable library of cryptographic commands and encryption algorithms. An encryption processor is included to perform key and data encryption, as well as a high performance hash processor and a public key accelerator.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

CRYPTOGRAPHIC CO-PROCESSOR

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is based on U.S. Provisional Patent Application Serial Nos. 60/059,082, 60/059,839, 60/059,840, 60/059,841, 60/059,842, 60/059,843, 60/059,844, 60/059,845 and 60/059,847, each of which was filed on September 16, 1997, the disclosures of which are incorporated herein by reference.

COPYRIGHT NOTICE

BACKGROUND OF THE INVENTION

Field Of The Invention

The present invention relates generally to a secure communication platform on an integrated circuit, and more particularly relates to a digital signal processor (DSP) with embedded encryption security features.

Description Of The Prior Art

Digital signal processors (DSPs) are widely used in devices such as modems, cellular telephones and facsimiles. With an increase in digital communications, data transmission security has become an issue in numerous DSP applications. A standard DSP is not capable of providing data transmission security; thus, additional hardware and software are required.

Security for digital communications is available on various integrated circuits. The integrated circuit security features include hardware implemented encryption algorithms such as the Data Encryption Standard (DES), Hash function algorithms and hardware implemented public key accelerators. The availability of this hardware makes it possible to provide security for distributed computing; however, no hardware implemented encryption algorithms have been known to be incorporated in a DSP.

Software encryption algorithms have also been developed to provide security for distributed computing. One commonly used encryption algorithm is the Data Encryption Standard (DES). DES is a block cipher which operates on 64-bit blocks of data and employs a 56-bit key. Another commonly used standard is the Digital Signature Algorithm (DSA). The DSA standard employs an irreversible public key system. These algorithms and more are part of the public domain and are available on the Internet.

Hash function algorithms are used to compute digital signatures and for other cryptographic purposes. One Hash function algorithm is the U.S. government's Secure Hash Algorithm (SHA-1).

Another security standard commonly used is the Internet Protocol Security Standard (IPsec). This standard provides security when communicating across the Internet. The standard requires DES to encrypt an Internet Protocol data packet, SHA-1 for authentication, and a public key algorithm for hand-shaking.

Since the IPsec standard requires different encryption algorithms, a software library is usually created so that a desired algorithm may be accessed when needed. Security systems employing encryption libraries are software implemented and designed specifically to run on the user's processor hardware.

Digital communication systems are not generally designed with security hardware. In most systems, security is achieved by software, such as described above, which is not entirely secure because there is no security hardware to block access to the security software by an intruder. Another problem associated with software encryption algorithms is that some of the software encryption algorithms run slower than when hardware implemented.

OBJECTS AND SUMMARY OF THE INVENTION

It is an object of the present invention to provide a digital signal processor with embedded security functions on a single integrated circuit.

It is another object of the present invention to provide a secure communications platform that can implement a user's application and dedicate cryptographic resources to encryption and decryption requests on demand.

It is another object of the present invention to provide an increase in encryption security through hardware implementations.

It is another object of the present invention to provide a security co-processor for high speed networking products such as routers, switches and hubs.

A cryptographic co-processor constructed in accordance with one form of the present invention includes a processor having encryption circuits built into it. The processor is capable of processing various applications, such as modem and networking applications. The encryption circuits and firmware make it possible to add security to the various processing applications. Hardware such as encryption and hash circuits are provided and structured to work together to provided accelerated encryption/decryption capabilities. A memory is programmed with cryptographic algorithms that support various encryption/decryption techniques. The cryptographic co-processor is structured so that a manufacturer of data communication products could substitute a current processor with the cryptographic co-processor and receive encryption capabilities with little modification to the existing product.

Since DSP's are the building block of many communication systems, a secured DSP with universal security features that may be selected by the manufacturer of the equipment in which the DSP forms part of would have far ranging benefits.

The benefits of a universal cryptographic co-processor (e.g., DSP) is that it can perform standard processor functions and standard encryption functions with no peripheral hardware or cryptographic software. Because the cryptographic co-

processor is implemented on a standard processor platform (i.e., substrate or monolithic chip), the processor that is being used in a manufacturer's product can be substituted with the cryptographic co-processor with little or no modification to the original design. The manufactured product incorporating the secure, universal co-processor now has encryption capabilities along with the original processor capabilities.

A preferred form of the cryptographic co-processor, as well as other embodiments, objects, features and advantages of this invention, will be apparent from the following detailed description of illustrated embodiments, which is to be read in connection with the accompanying drawing.

BRIEF DESCRIPTION OF THE DRAWING

Figure 1 is a block diagram of the cryptographic co-processor formed in accordance with the present invention.

Figure 1A is a block diagram similar to Figure 1 showing another view of the cryptographic co-processor of the present invention.

Figure 2 is a block diagram of the program memory preferably used in the co-processor of the present invention for MMAP=0.

Figure 3 is a block diagram of the program memory preferably used in the co-processor of the present invention for MMAP=1.

Figure 4 is a block diagram of the data memory preferably used in the co-processor of the present invention.

Figure 5 is a block diagram of the PCI memory preferably used in the co-processor of the present invention.

Figure 6 is a block diagram of the DMA subsystem preferably used in the co-processor of the present invention.

Figure 7 is a block diagram of the DSP "local" memory preferably used in the co-processor of the present invention.

Figure 8 is a flow chart showing the DMA control preferably used in the co-processor of the present invention.

Figure 9 is a block diagram of the hash/encrypt circuit preferably used in the co-processor of the present invention.

Figure 10 is a block diagram of the interrupt controller preferably used in the co-processor of the present invention.

Figure 11 is a block diagram of the CGX software interface preferably used in the co-processor of the present invention.

Figure 12 is a block diagram illustrating the layers of software preferably used in the co-processor of the present invention.

Figure 13 is a block diagram of the CGX overlay interface preferably used in the co-processor of the present invention.

Figure 14 is a block diagram illustrating the hierarchical interface of the CGX kernel cryptographic service preferably used in the co-processor of the present invention.

Figure 15 is a functional state diagram illustrating the CGX kernel preferably used in the co-processor of the present invention.

Figure 16 is a functional tree diagram showing the KEK hierarchy preferably used in the co-processor of the present invention.

Figure 17 is a block diagram of a symmetric key weakening algorithm preferably used in the co-processor of the present invention.

Figure 18 is a functional tree diagram showing the symmetric key hierarchy preferably used in the co-processor of the present invention.

Figure 19 is a portion of a computer program defining the PCDB data type preferably used in the co-processor of the present invention.

Figure 20 is a block diagram of the kernel block preferably used in the co-processor of the present invention.

Figure 21 is a portion of a computer program defining the kernel block preferably used in the co-processor of the present invention.

Figure 22 is a portion of a computer program defining the command block preferably used in the co-processor of the present invention.

Figure 23 is a portion of a computer program defining the secret key object preferably used in the co-processor of the present invention.

Figure 24 is a portion of a computer program defining the public keyset preferably used in the co-processor of the present invention.

Figure 25 is a portion of a computer program defining the Diffie-Hellman public keyset preferably used in the co-processor of the present invention.

Figure 26 is a portion of a computer program defining the RSA public keyset preferably used in the co-processor of the present invention.

Figure 27 is a portion of a computer program defining the DSA public keyset preferably used in the co-processor of the present invention.

Figure 28 is a portion of a computer program defining the DSA digital signature preferably used in the co-processor of the present invention.

Figure 29 is a portion of a computer program defining the DSA seed key preferably used in the co-processor of the present invention.

Figure 30 is a portion of a computer program defining the key cache register data type preferably used in the co-processor of the present invention.

Figure 31 is a portion of a computer program defining the symmetrical encryption context store preferably used in the co-processor of the present invention.

Figure 32 is a portion of a computer program defining the one-way hash context store preferably used in the co-processor of the present invention.

Figure 33 is a portion of a computer program defining one example of the CGX wrap code and command interface preferably used in the co-processor of the present invention.

Figure 34 is a portion of a computer program defining the CGX overlay table preferably used in the co-processor of the present invention.

Figure 35 is a portion of a computer program defining the KCS object preferably used in the co-processor of the present invention.

WHAT IS CLAIMED IS:

1. A cryptographic co-processor comprising:

a processing unit for processing data;

a read only memory electronically linked to the processing unit and including a masked programmed cryptographic library of encryption algorithms; and

an encryption processor for encrypting data, the encryption processor and the processing unit being situated on the same platform.

2. A cryptographic co-processor comprising:

a digital signal processor having a memory associated therewith;

a cryptographic library having user selectable encryption algorithms masked programmed into the memory; and

security hardware embedded within the digital signal processor.

3. A cryptographic co-processor as defined by Claim 2, wherein the security hardware includes an encryption circuit, the encryption circuit electronically linked to the digital signal processor and performing encryption and decryption functions.

4. A cryptographic co-processor as defined by Claim 2, wherein the security hardware includes a HASH circuit, the HASH circuit being electronically linked to the digital signal processor and performing HASH functions.

5. A cryptographic co-processor as defined by Claim 2, wherein the security hardware includes a public key accelerator circuit, the public key accelerator circuit being electronically linked to the digital signal processor and performing arithmetic functions.

6. A cryptographic co-processor as defined by Claim 2, wherein the security hardware includes a random number generator, the random number

generator being electronically linked to the digital signal processor and generating random numbers used for encryption purposes.

7. A cryptographic co-processor as defined by Claim 1, which further comprises:

a laser trimmed memory, the laser trimmed memory being electronically linked to the processing unit and having stored therein a master key used by the cryptographic co-processor, the master key being programmed into the memory by laser trimming.

8. A cryptographic service software embodied in at least one of a hard disc, a floppy disc and a read-only memory (ROM), the cryptographic service software electronically communicating and being compatible with a standard operating system of a computer, the operating system having an application space and a kernel space, the cryptographic service software performing cryptographic services at the kernel space of the operating system, which comprises:

a generic layer including a kernel space level program interface; and

a cryptographic service module having a library of encryption algorithms, the module electronically communicating and cooperating with the program interface.

9. In combination, a first cryptographic service software embodied in at least one of a floppy disc and a read only memory (ROM), the first cryptographic service software electronically communicating and being compatible with a standard operating system of a computer, the operating system having an application space and a kernel space, the cryptographic service software performing cryptographic services at the application space of the operating system, the cryptographic service software comprising an application program interface, and a first cryptographic service module, the first cryptographic service module having a library of encryption algorithms, the first cryptographic service module electronically communicating and cooperating with the application program interface; and

a second cryptographic service software embodied in at least one of a floppy disc and a read only memory (ROM), the second cryptographic service software electronically communicating and being compatible with the operating system of the computer, the second cryptographic service software performing cryptographic services at the kernel space of the operating system, the second cryptographic service software including a kernel space level program interface, and a second cryptographic service module, the second cryptographic service module having a library of encryption algorithms, the second cryptographic service module electronically communicating and cooperating with the kernel space level program interface.

10. A computer having an operating system, the operating system including an application space and a kernel space, which comprises:

at least one security enabled kernel engine situated in the kernel space of the operating system;

cryptographic service software, the cryptographic service software being situated at least in the kernel space of the operating system, the cryptographic service software including at least one program interface electronically communicating with the at least security enabled kernel engine, and at least one cryptographic service module electronically communicating with the at least one program interface, the at least one cryptographic service module including a library of encryption algorithms.

11. A computer as defined by Claim 10, which further comprises:

a cryptographic co-processor, the cryptographic co-processor including a memory and a second library of encryption algorithms mask-programmed into the memory, the co-processor electronically communicating with the at least one program interface of the cryptographic service software.

12. A method of securely communicating between an application program and a secure kernel of an integrated circuit, the secure kernel having stored therein cryptographic algorithms, the integrated circuit further having a register, a command block memory and a kernel block memory, which comprises the steps of:

storing in the register of the integrated circuit the address of the kernel block memory;

transferring the kernel block memory address stored in the register to the secure kernel;

reading by the secure kernel the contents of the kernel block memory at the transferred address, the contents of the kernel block memory at the transferred address containing at least one pointer address corresponding to a memory location in the command block memory; and

fetching by the secure kernel the contents of the memory location of the command block memory corresponding to the at least one pointer address, the contents of the memory location including at least one of a command and an argument.

13. A hardware secure memory area, which comprises:

a main communication bus;

a plurality of secondary communication buses;

a plurality of bus transceivers coupling the plurality of secondary communication buses to the main communication bus; and

a plurality of memory circuits coupled to the plurality of communication buses, each bus transceiver selectively isolating a secondary communication bus to which the bus transceiver is associated from the main communication bus and selectively causing communication between the associated secondary communication bus and the main communication bus.

14. A hardware secure memory area, which comprises:

a main communication bus;

a first bus transceiver coupled to the main communication bus;

a second bus transceiver coupled to the main communication bus;

a third bus transceiver coupled to the main communication bus;

a key communication bus coupled to the first bus transceiver;

a key cache coupled to the key communication bus for writing and reading keys;

a key random access memory coupled to the key communication bus for writing and reading cryptographic operations and keys;

a processor memory for writing and reading cryptographic algorithms, operations and keys;

an external memory communication bus coupled to the second bus transceiver;

an external memory coupled to the external memory communication bus for writing and reading application programs and commands;

a cryptographic algorithm communication bus coupled to the third bus transceiver;

a scratch memory coupled to the cryptographic algorithm communication bus for writing and reading cryptographic calculations; and

a memory coupled to the cryptographic algorithm communication bus for storing cryptographic algorithms.

15. A hardware secure memory area, which comprises:

a main communication bus;

a plurality of bus transceivers coupled to the main communication bus for controlling access to and from the main communication bus;

a plurality of secondary communication buses coupled to the bus transceivers; and

a plurality of memory circuits coupled to the plurality of secondary communication buses.

16. A method of expanding a protected memory area of a secure kernel into an unprotected memory area in an integrated circuit, which comprises the steps of:

initializing the secure kernel including the step of sending thereto a command signal having a starting memory block address and the number of memory blocks to be protected;

reading the command signal by the secure kernel;

writing the starting memory block address and the number of memory blocks into a memory of the secure kernel; and

allocating the starting memory block address and the number of memory blocks as protected memory.

17. A method of expanding a protected memory area of a secure kernel into an unprotected memory area in an integrated circuit, which comprises the steps of:

initializing the secure kernel by a command generated by an application program, the command causing arguments attached to the command to be vectored into the secure kernel, the arguments including pointer data and the number of protected memory blocks requested, the pointer data containing the starting memory address of the protected memory requested;

reading the arguments by the secure kernel;

writing into a data memory reserve register a code corresponding to the number of memory blocks to be protected; and

allocating the number of memory blocks requested to be protected as protected memory starting at the starting memory block address contained in the pointer data.

18. A method of expanding a secured memory into an unprotected memory to define an additional secured memory area, the secured memory being expanded to accommodate storage of an extended code, the extended code being initially stored in the unprotected memory in a location which will become the additional secured memory area, the secured memory and the unprotected memory forming parts of an integrated circuit, the integrated circuit having a serial number stored in a memory thereof, which comprises the steps of:

retrieving by an authorizing party the serial number stored in the memory of the integrated circuit and the extended code proposed to be stored in the requested expanded secured memory;

verifying by the authorizing party whether the extended code is acceptable to be stored in the expanded secured memory of the integrated circuit;

generating a token signal by the authorizing party and communicating the token signal to the integrated circuit, the token signal including at least the digital signature of the extended code, as computed by the authorizing party,

receiving the token signal by the integrated circuit and parsing the token signal to separate the digital signature of the authorizing party;

verifying by the integrated circuit the digital signature of the authorizing party parsed from the token signal which, if verified, indicates that the authorizing party authorizes the expansion of the secured memory by the integrated circuit; and

invoking by the integrated circuit a command to expand the secured memory so that the additional secured memory area now encompasses the location of the unprotected memory where the extended code is stored.

19. A method of expanding a secured memory as defined by Claim 18, wherein the token signal further includes the serial number of the integrated circuit; and wherein the method further comprises the steps of:

parsing by the integrated circuit from the token signal returned by the authorizing party the serial number of the integrated circuit; and

verifying by the integrated circuit the serial number parsed by the token signal by comparing the parsed serial number with the serial number stored in the non-volatile memory of the integrated circuit.

20. A method of expanding a secure kernel memory area into an unprotected memory, while providing security to the unprotected memory area and validating an extended code, the secure kernel memory and the unprotected memory forming part of an integrated circuit, comprising the steps of:

reading an integrated circuit serial number;

communicating the serial number to a manufacturer;

loading code into a memory;

loading a token into a memory, the token includes the ICs serial number;

digitally signing the extended code by a trusted authority;

copying the extended code into the ICs unprotected memory;

invoking a command to add an extension to the secure kernel memory;

specifying which memory blocks are to be acquired from the unprotected memory area;

disabling further program access to the unprotected memory area;

verifying the digital signature computed over the extended code is authentic;

locking the extended code into the protected memory area; and

permitting the secure kernel to make calls to the protected memory area.

21. A method of reconfiguring the functionality of an integrated circuit, the integrated circuit having a serial number stored in a memory thereof, which comprises the steps of:

retrieving the serial number from the integrated circuit;

transmitting a token request signal by the integrated circuit to an authorizing party, the token request signal including at least the serial number of the integrated circuit;

transmitting a return token signal from the authorizing party to the integrated circuit, the return token signal including at least the serial number of the integrated circuit, a reconfiguration code to be used for reconfiguring the integrated circuit and a digital signature of the authorizing party;

parsing the return token signal by the integrated circuit to extract the serial number, reconfiguration code and digital signature;

verifying the serial number by the integrated circuit by comparing the parsed serial number from the return token signal with the serial number stored in the memory of the integrated circuit;

verifying the digital signature of the authorizing party by the integrated circuit using a public key stored in the memory of the integrated circuit;

storing the parsed reconfiguration code from the return token signal in a memory of the integrated circuit; and

reconfiguring the integrated circuit in accordance with the reconfiguration code.

22. A method of reconfiguring the functionality of an integrated circuit, the integrated circuit having a serial number stored in a memory thereof, which comprises the steps of:

retrieving the serial number from the integrated circuit;

transmitting by the integrated circuit a token request signal to an authorizing party, the token request signal including the serial number and a reconfiguration code;

transmitting a return token signal from the authorizing party to the integrated circuit, the return token signal including the serial number of the integrated circuit, the reconfiguration code and a digital signature of the authorizing party;

parsing the return token signal by the integrated circuit to extract the serial number, reconfiguration code and digital signature;

verifying the serial number by the integrated circuit by comparing the parsed serial number from the return token signal with the serial number stored in the memory of the integrated circuit;

verifying the digital signature of the authorizing party by the integrated circuit using a public key stored in a memory of the integrated circuit;

storing the reconfiguration code in a memory of the integrated circuit; and

reconfiguring the integrated circuit in accordance with the reconfiguration code.

23. A method of generating a recovery key encryption key (RKEK) in a secure manner by an integrated circuit and a key recovery escrow agent, which comprises the steps of:

generating by the integrated circuit a first number having a private component and a public component;

generating by the escrow agent a second number having a private component and a public component;

providing the public component of the first number to the escrow agent;

providing the public component of the second number to the integrated circuit;

conducting a mathematical operation by the integrated circuit using the private component of the first number, and the public component of the second number to create the RKEK; and

conducting a mathematical operation by the escrow agent using the private component of the second number, and the public component of the first number to create the RKEK.

24. A method of generating a recovery key encryption key (RKEK) in a secure manner by an integrated circuit and a key recovery escrow agent, the integrated circuit having a unique serial number stored in a memory of the integrated circuit, which comprises the steps of:

generating by the integrated circuit a first number having a private component and a public component;

generating by the escrow agent a second number having a private component and a public component;

retrieving by a third party the serial number of the integrated circuit and comparing the serial number with a serial number stored in a memory of the third party to verify the identity of the integrated circuit;

generating by the third party a message containing at least a digital signature of the third party authorizing the generation of the RKEK and communicating the message to the integrated circuit;

providing the public component of the second number to the integrated circuit; and

conducting a Diffie-Hellman modulo-exponentiation mathematical operation by the integrated circuit using the private component of the first number, and the public component of the second number to create the RKEK.

25. A method of generating a recovery key encryption key (RKEK) as defined by Claim 24, wherein the message generated by the third party and communicated to the integrated circuit further includes the serial number of the integrated circuit, and wherein the method further comprises the step of:

verifying by the integrated circuit the accuracy of the serial number included in the message by comparing the serial number of the message with the serial number stored in the memory of the integrated circuit.

26. A method of generating a recovery key encryption key (RKEK) as defined by Claim 25, which further comprises the step of:

verifying by the integrated circuit the accuracy of the digital signature of the third party contained in the method..

27. A method of generating a recovery key encryption key (RKEK) as defined by Claim 26, which further comprises the steps of:

providing the public component of the first number to the escrow agent; and

conducting a Diffie-Hellman modulo-exponentiation mathematical operation by the escrow agent using the private component of the second number, and the public component of the first number to create the RKEK.

28. A controller circuit for switching between a user mode and a kernel mode in a processor comprising;

a processor;

a program counter electrically connected to the processor for monitoring program fetch addresses;

a kernel program fetch supervisor circuit having a predetermined address value stored within, electrically connected to the program counter for comparing the address in the program counter to the predetermined address value stored within;

a program memory electrically connected to the program counter;

a flip-flop circuit electrically connected to the kernel program fetch supervisor circuit for switching between setting a user mode bit and a kernel mode bit;

a kernel data fetch supervisor circuit electrically connected to the processor for comparing a data fetch address to a predetermined memory address range;

a data memory electrically connected to a processor data interface for storing data;

a first AND circuit coupled to the flip-flop and the kernel data fetch supervisor circuit for activating and deactivating a violation reset;

and a second AND circuit coupled to the first AND circuit and the kernel program fetch supervisor circuit for activating and deactivating the violation reset bit.

29. A method of monitoring and controlling program fetch addresses and data fetch addresses from a processor to control access to a protected memory comprising the steps of:

fetching a program opcode;

reading a program opcode address;

determining whether the program opcode address is fetched from one of a protected program memory address and an unprotected program memory address;

resetting the processor when the program opcode is fetched from the protected program memory address;

fetching a data operand when the program opcode address is fetched from the unprotected program memory address;

fetching a data operand and reading the data operand address;

determining whether the data operand address is fetched from one of a protected data memory address and an unprotected data memory address;

resetting the processor when the data operand is fetched from the protected data memory address;

calling a starting address of the protected program memory when the data operand address is fetched from the unprotected data memory;

fetching a second program opcode;

reading the second program opcode address;

determining whether the second program opcode address is fetched from one of a protected program memory address and an unprotected program memory address;

fetching a third program opcode when the second program opcode address is fetched from the unprotected memory address; and

fetching a second data operand when the second program opcode address is fetched from the protected memory address.

30. A method of managing the use of keys in cryptographic co-processor, which comprises the steps of:

selecting a key from one of a symmetrical key type and asymmetrical key type;

selecting a bit length from the selected key;

generating the key; and

representing the key in one of an external form and an internal form.

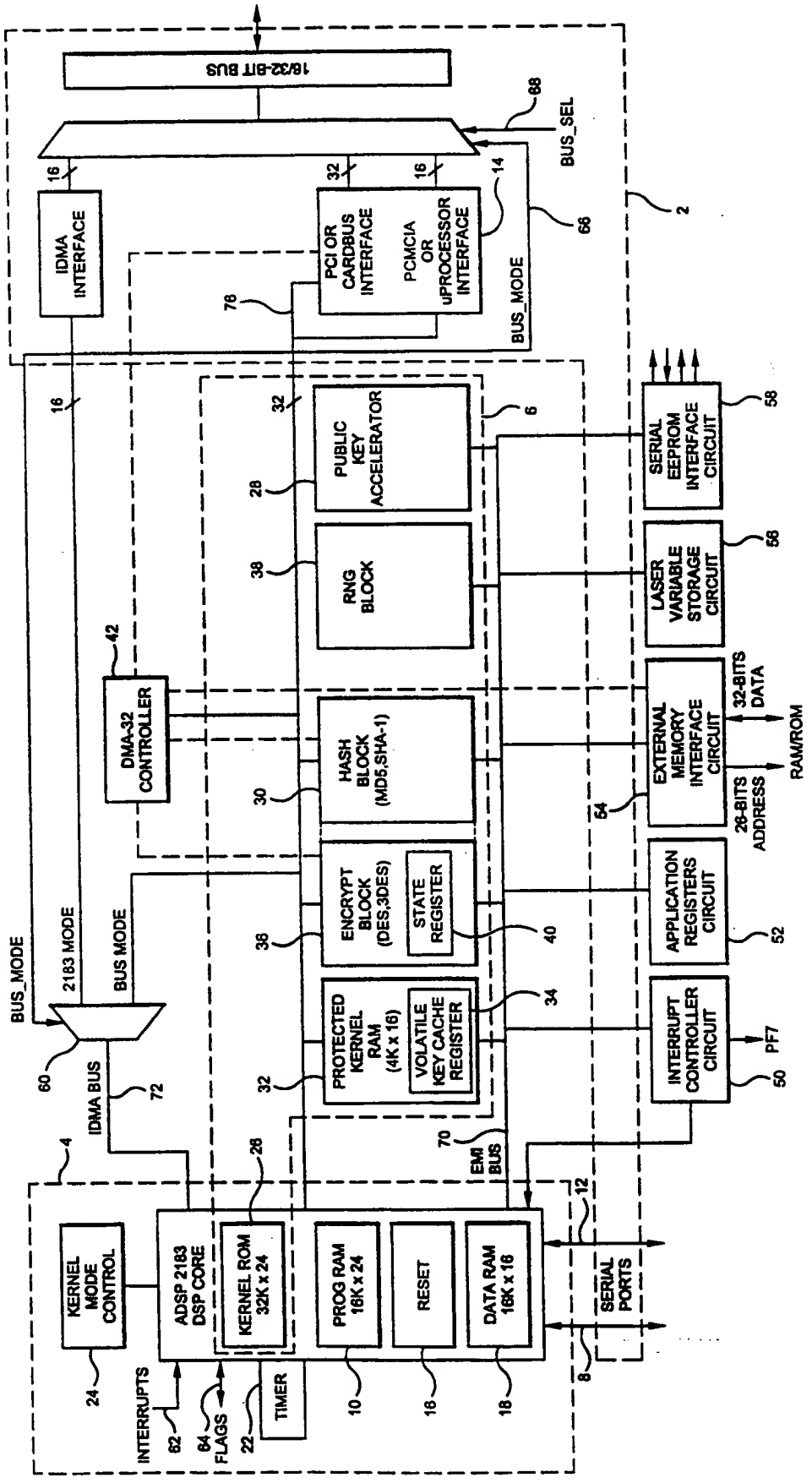

16/32-BIT BUS
BUS_SEL
68
IDMA INTERFACE
PCI OR CARDBUS INTERFACE
PCMCIA OR uPROCESSOR INTERFACE
BUS_MODE
66
2
76
14
PUBLIC KEY ACCELERATOR
RNG BLOCK
HASH BLOCK (MD5,SHA-1)
ENCRYPT BLOCK (DES,3DES)
STATE REGISTER
PROTECTED KERNEL RAM (4K x 16)
VOLATILE KEY CACHE REGISTER
DMA-32 CONTROLLER
SERIAL EEPROM INTERFACE CIRCUIT
LASER VARIABLE STORAGE CIRCUIT
EXTERNAL MEMORY INTERFACE CIRCUIT
APPLICATION REGISTERS CIRCUIT
INTERRUPT CONTROLLER CIRCUIT
32-BITS DATA
RAM/ROM
26-BITS ADDRESS
PF7
BUS_MODE
2183 MODE
BUS MODE
IDMA BUS
EMI BUS
KERNEL MODE CONTROL
ADSP 2183 DSP CORE
KERNEL ROM 32K x 24
PROG RAM 16K x 24
RESET
DATA RAM 16K x 16
SERIAL PORTS
TIMER
INTERRUPTS
FLAGS

FIG-1

FIG-1A CRYPTIC FUNCTIONAL BLOCK DIAGRAM

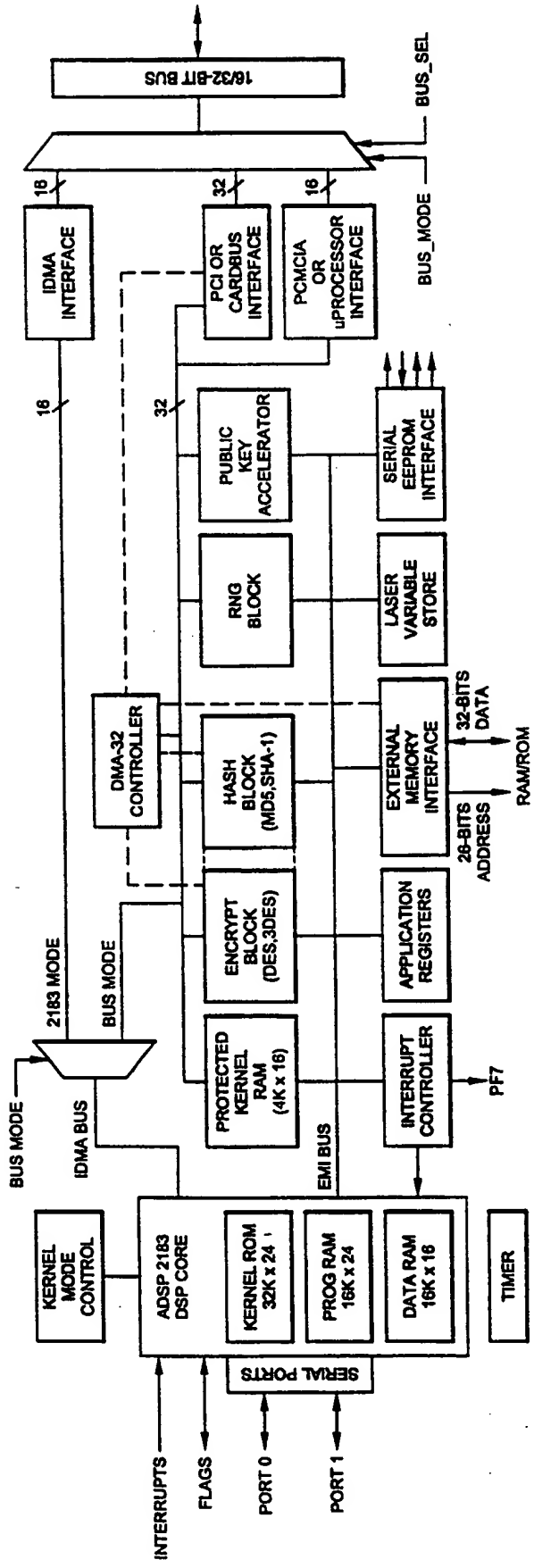


FIG-2 PROGRAM MEMORY (MMAP=0)

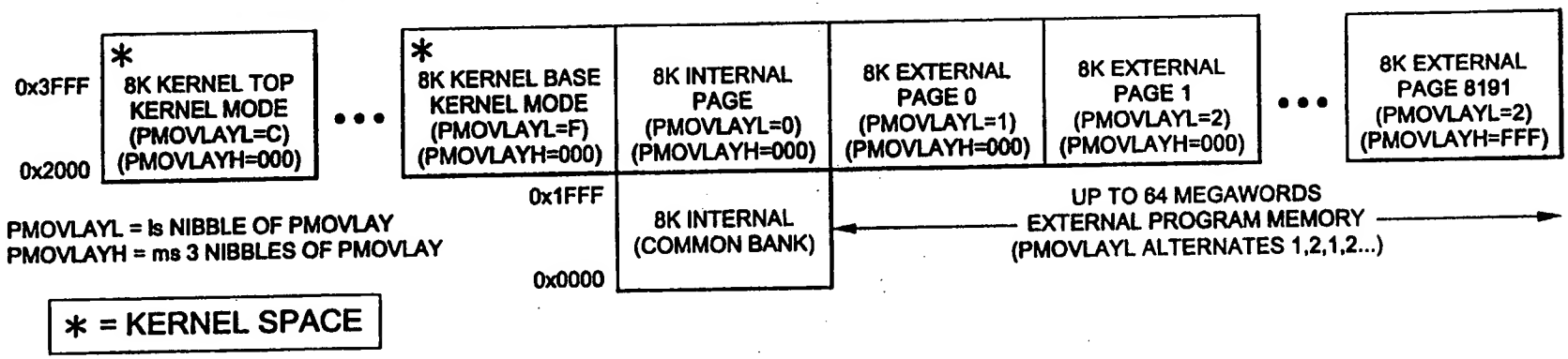


FIG-3 PROGRAM MEMORY (MMAP=1)

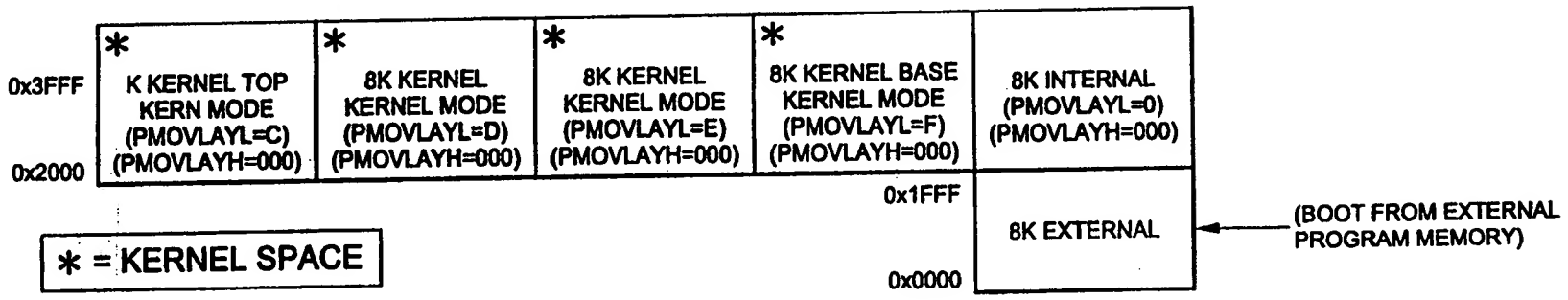


FIG-4 DATA MEMORY

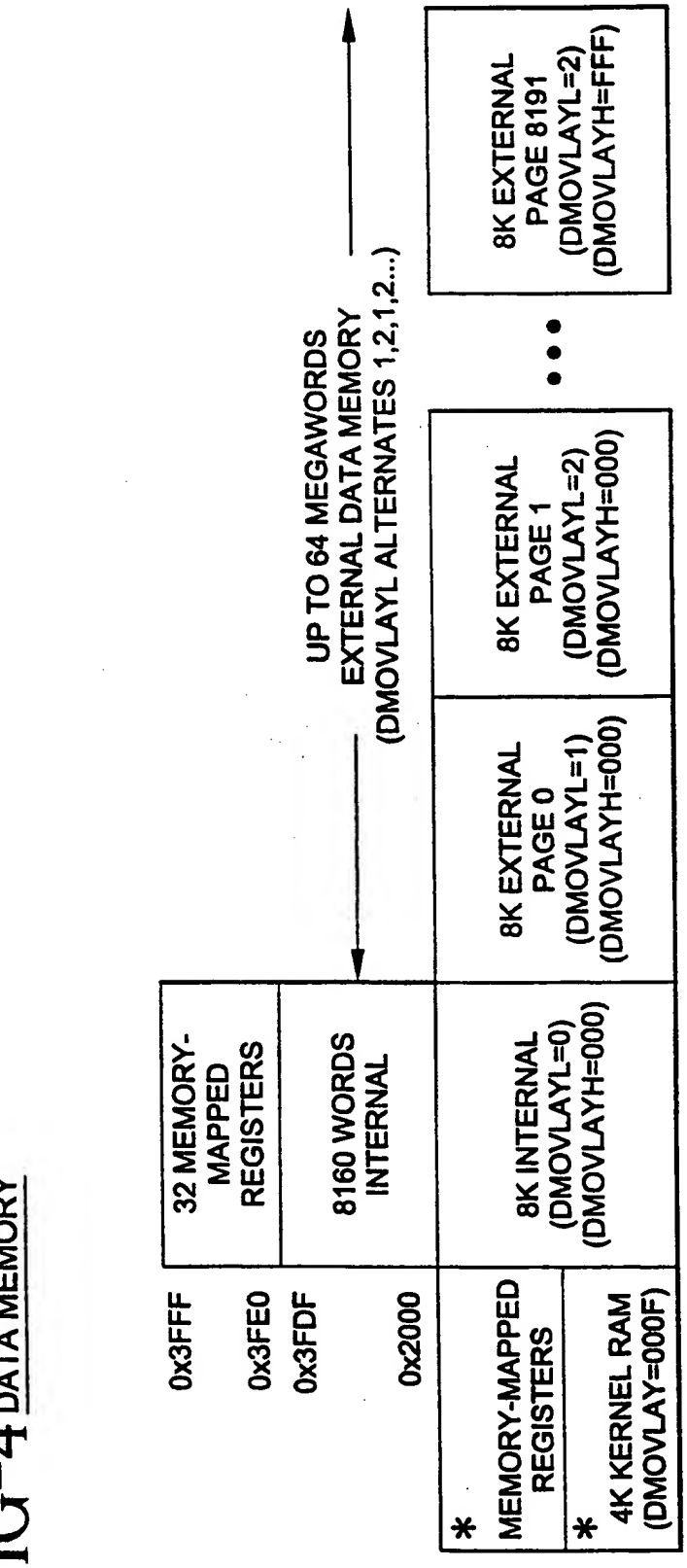
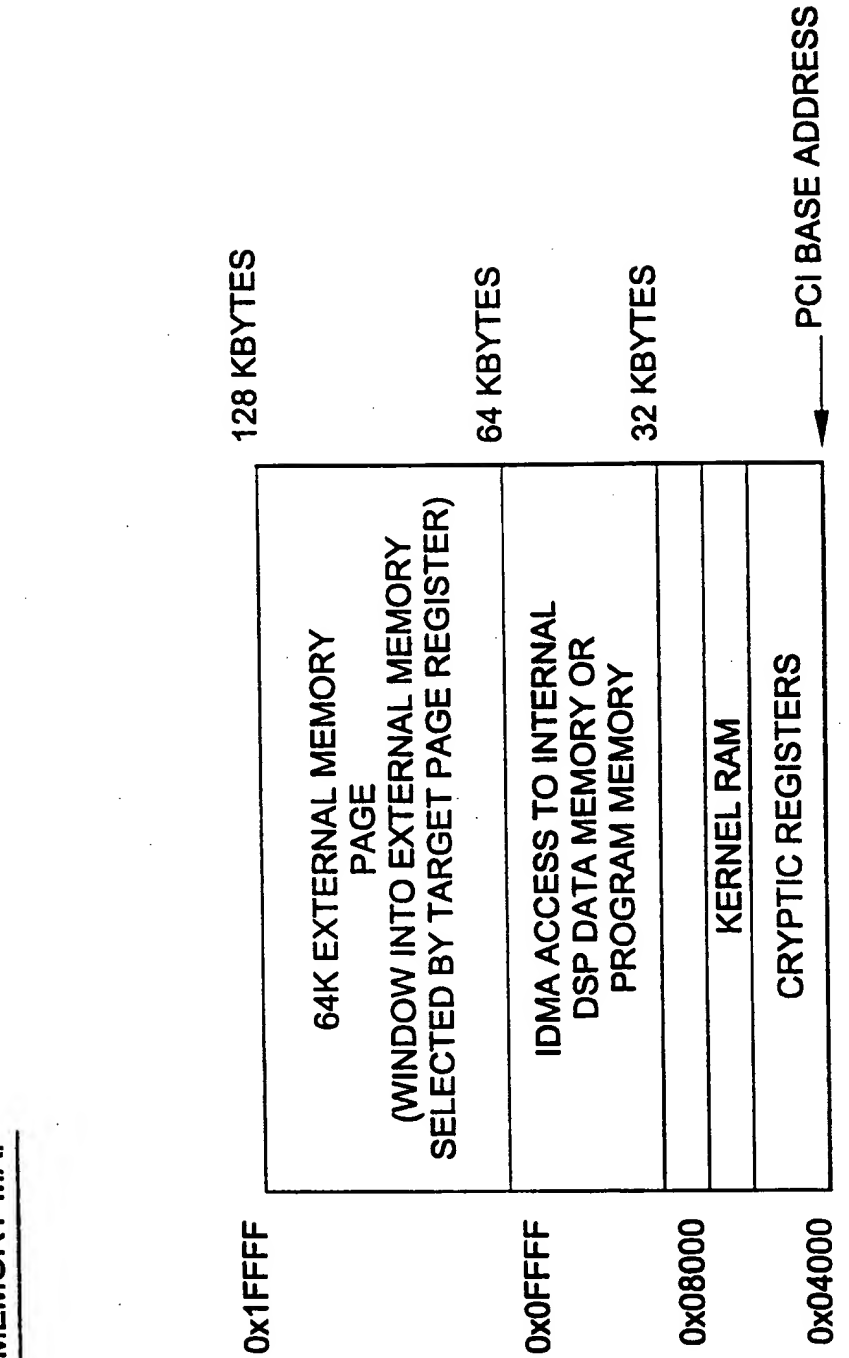


FIG-5 PCI MEMORY MAP



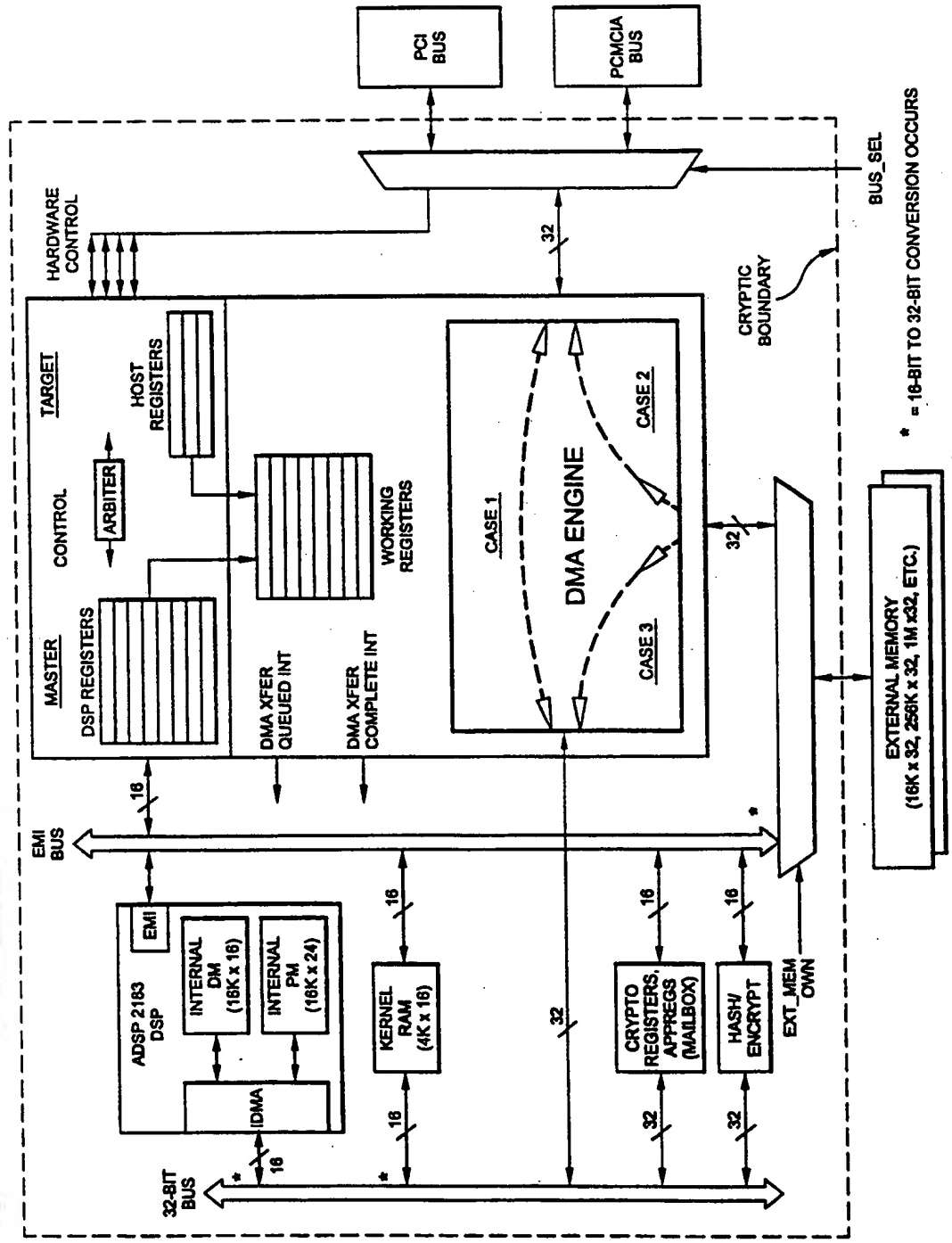


FIG-6 32-BIT DMA SUBSYSTEM

FIG-7 DSP "LOCAL" MEMORY MAP

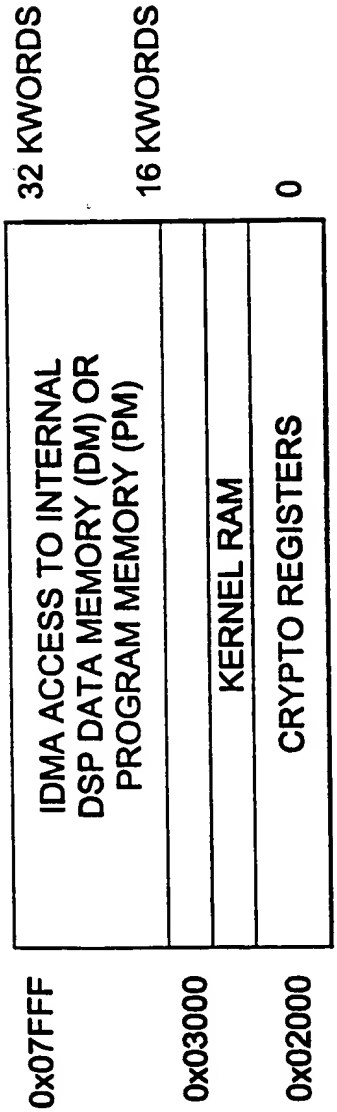


FIG-8 MASTER DMA FLOWCHART

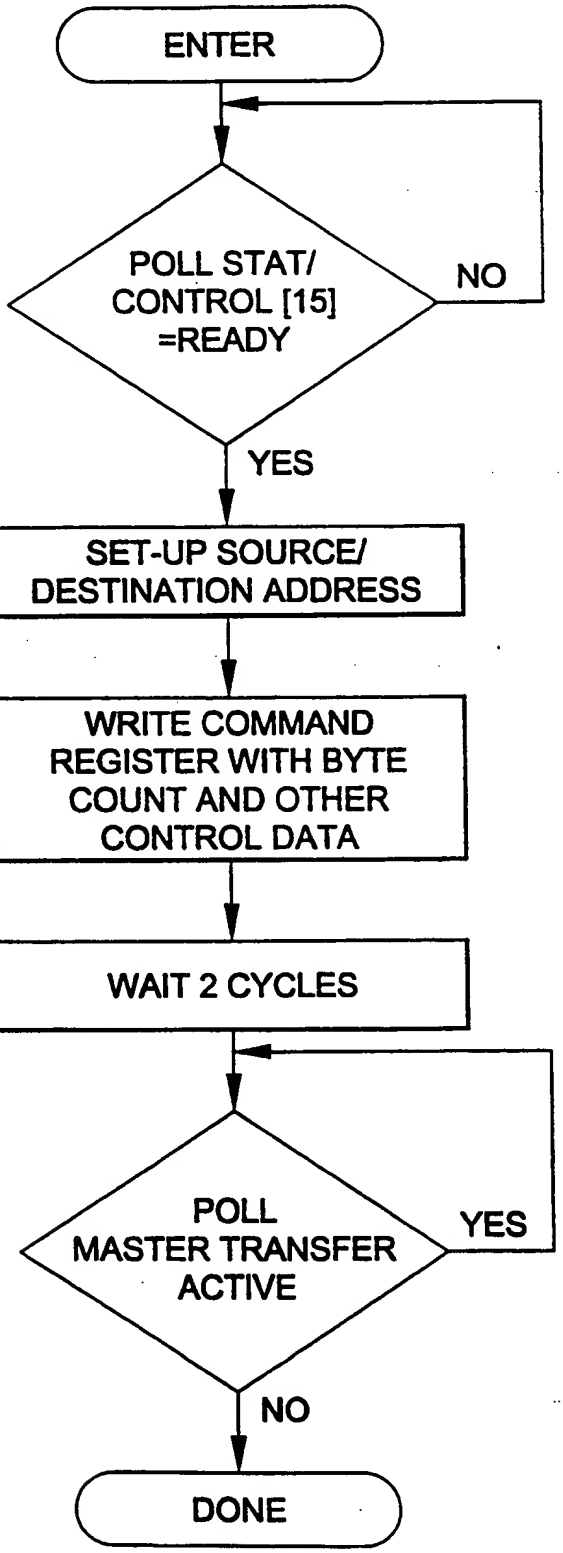


FIG-9 HASH/ENCRYPT FUNCTIONAL BLOCK DIAGRAM

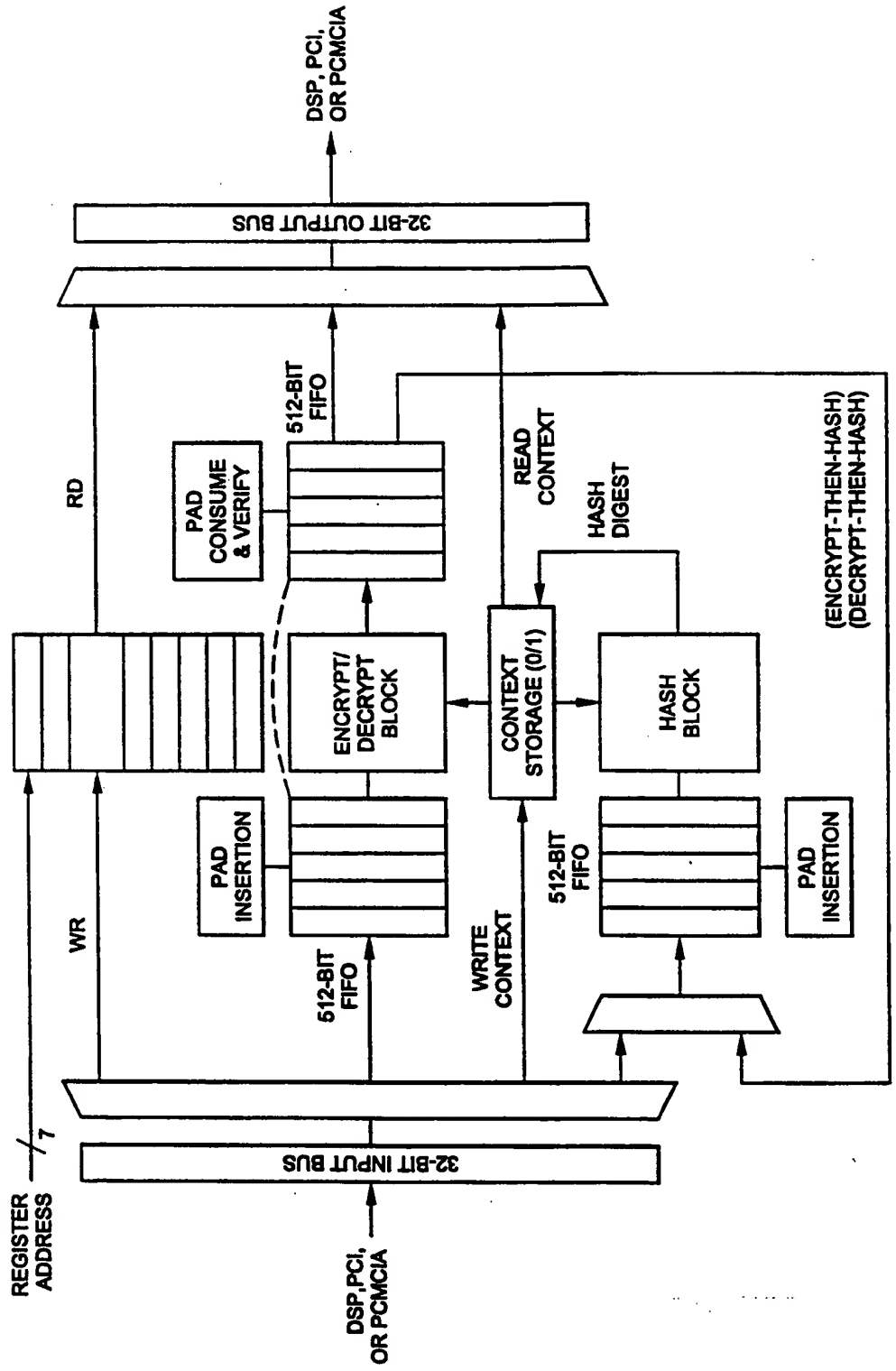


FIG-10 INTERRUPT CONTROLLER BLOCK DIAGRAM

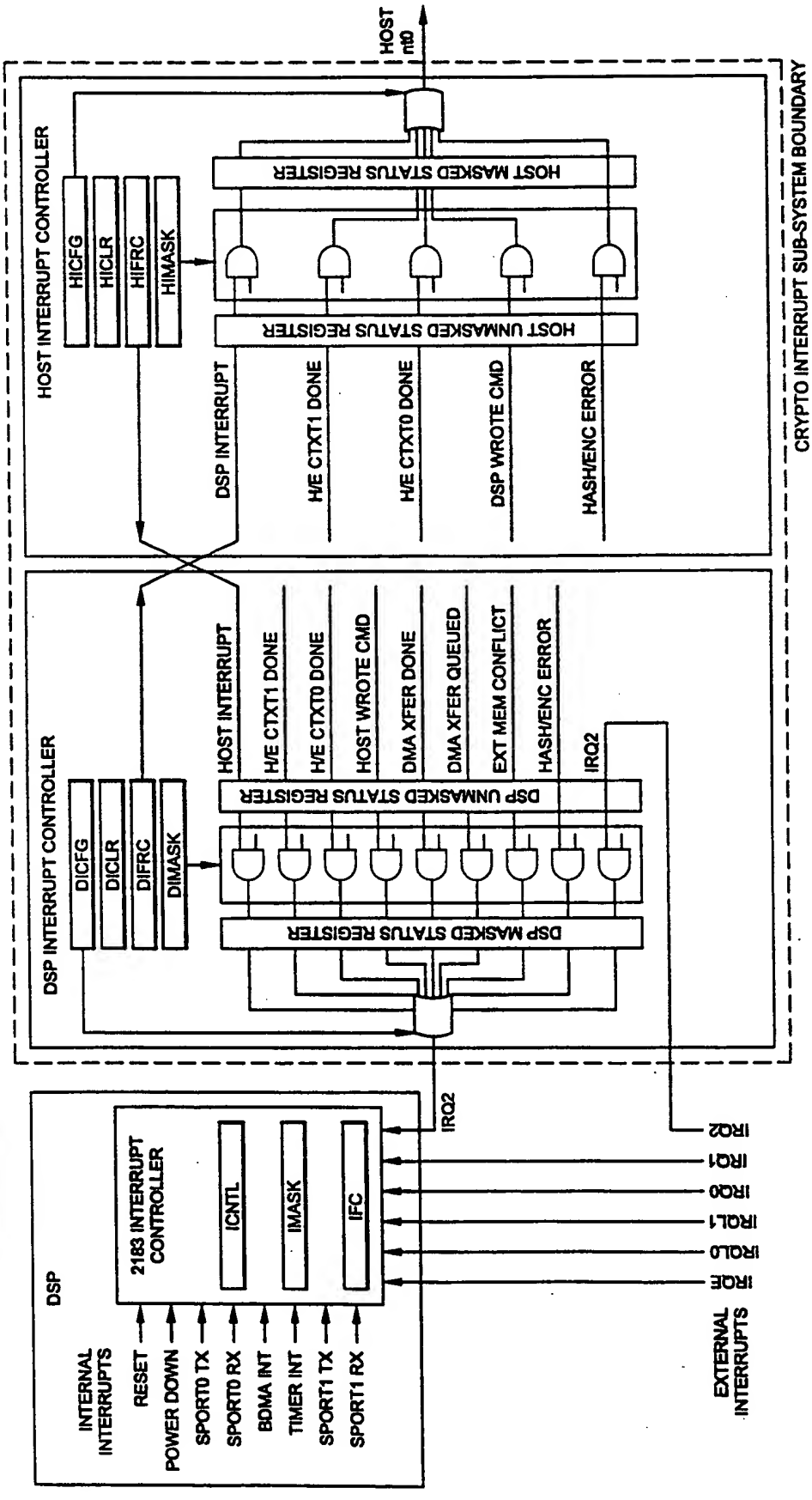


FIG-11 CGX SOFTWARE INTERFACE

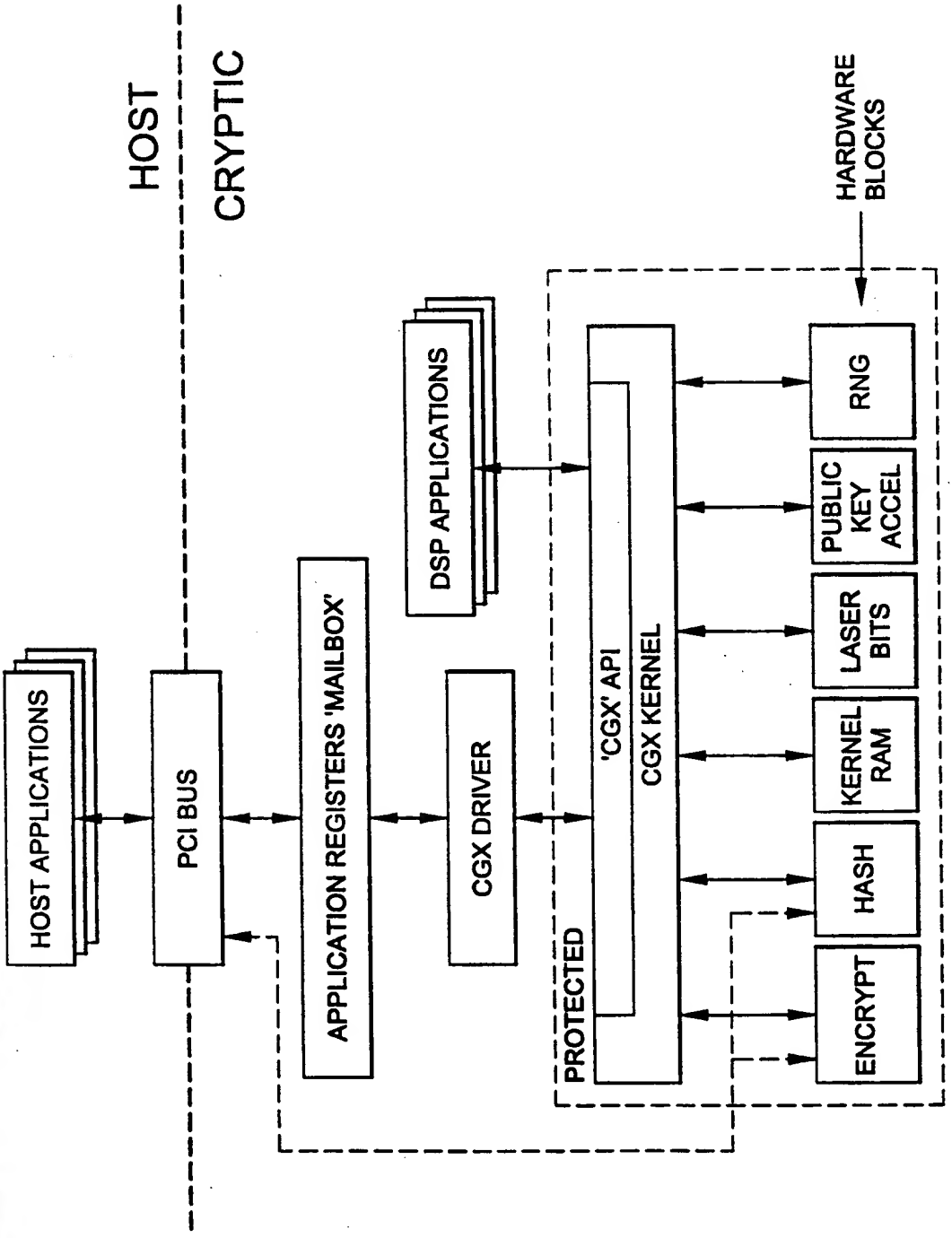


FIG-12 CRYPTIC SOFTWARE LAYERS

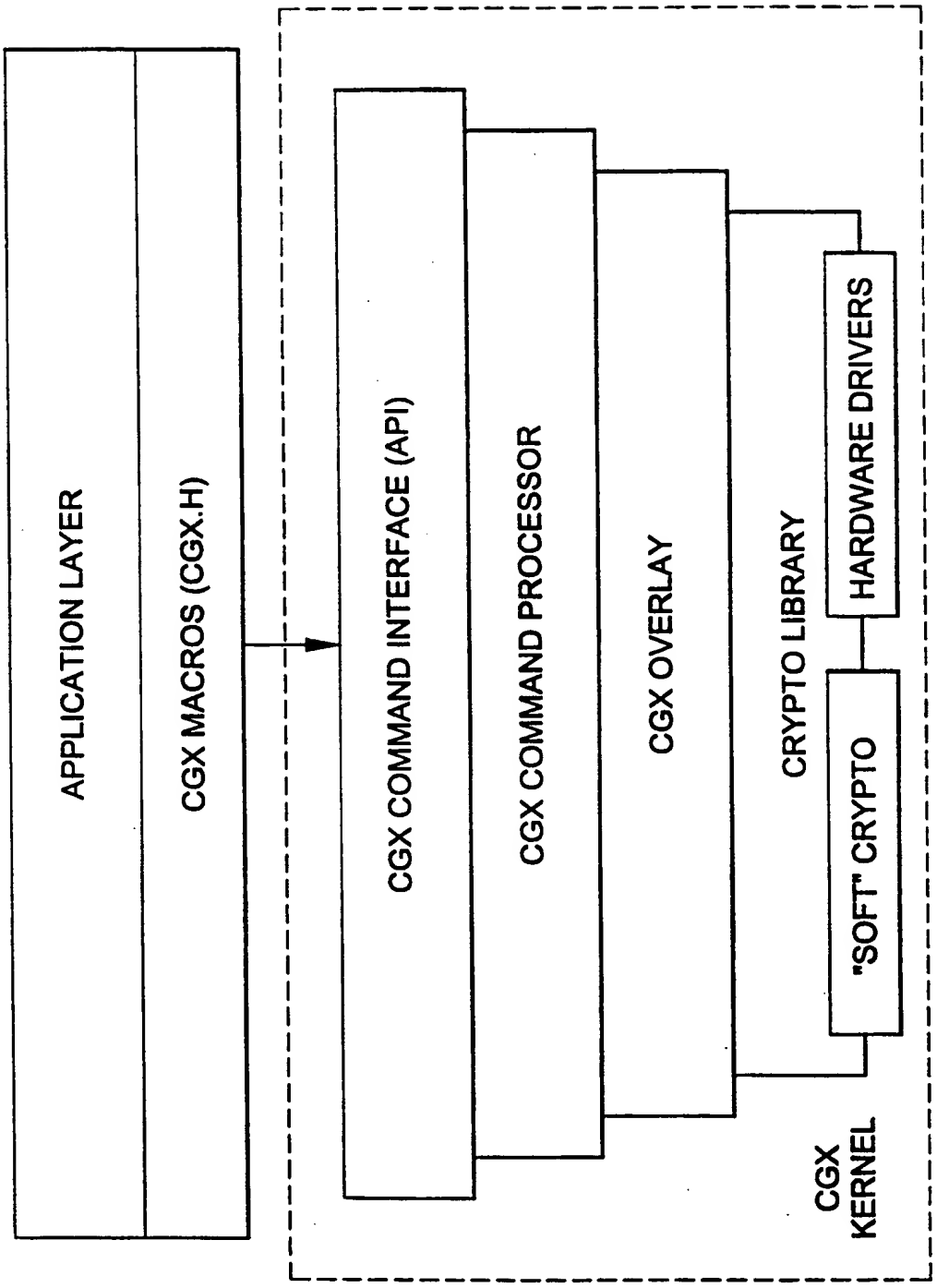


FIG-13 CGX OVERLAY INTERFACE

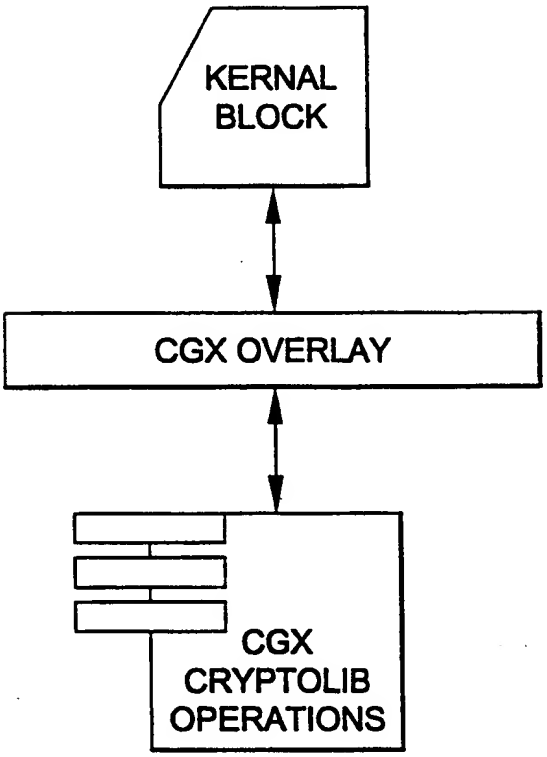
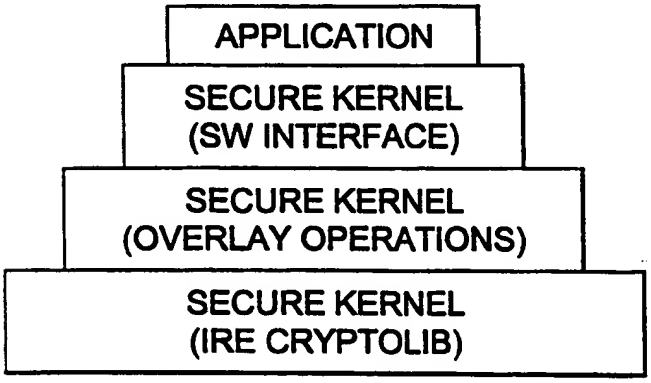


FIG-14 CGX KERNEL CRYPTOGRAPHIC SERVICE HIERARCHICAL INTERFACE



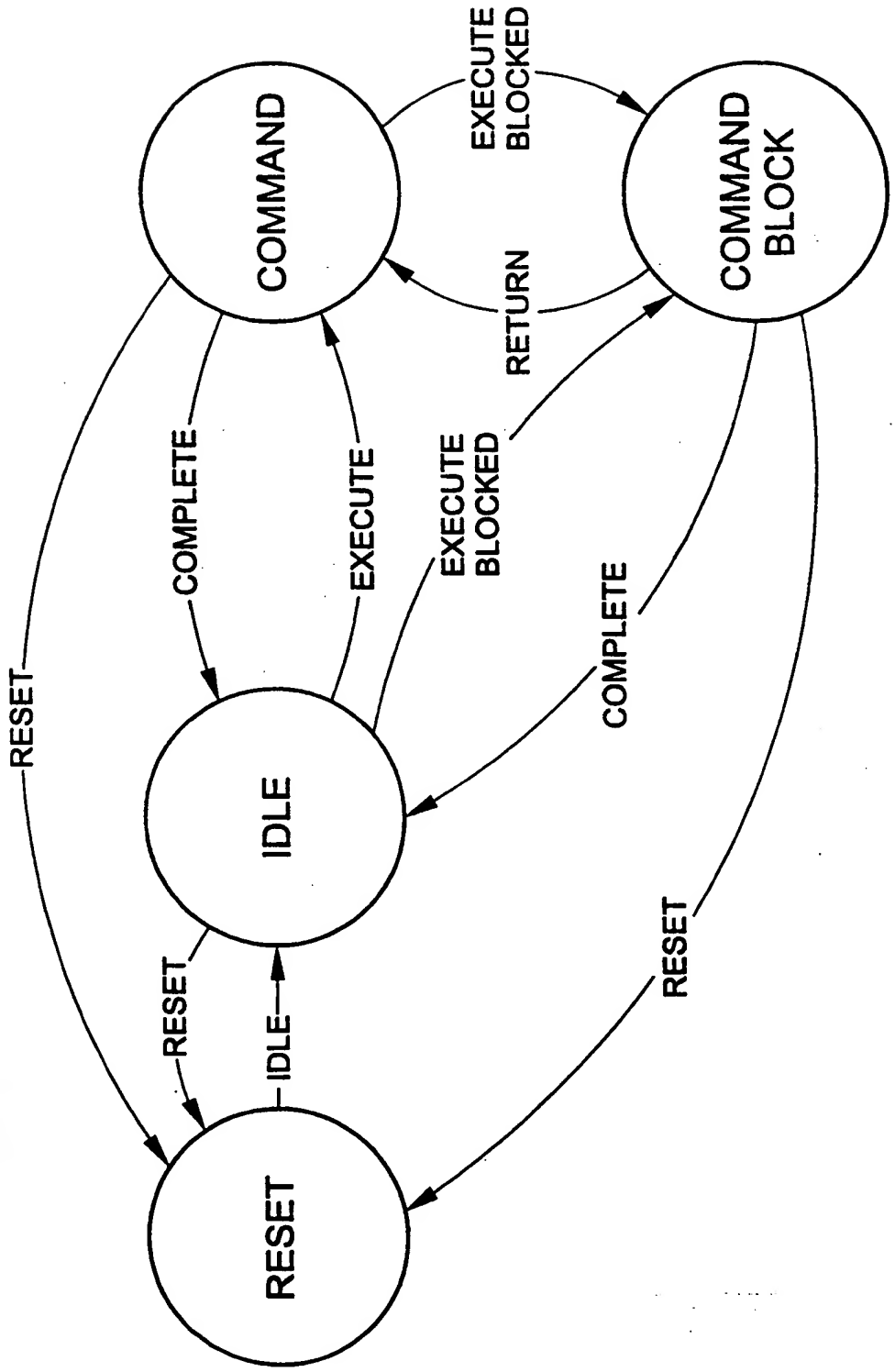


FIG-15 CGX KERNEL STATE DIAGRAM

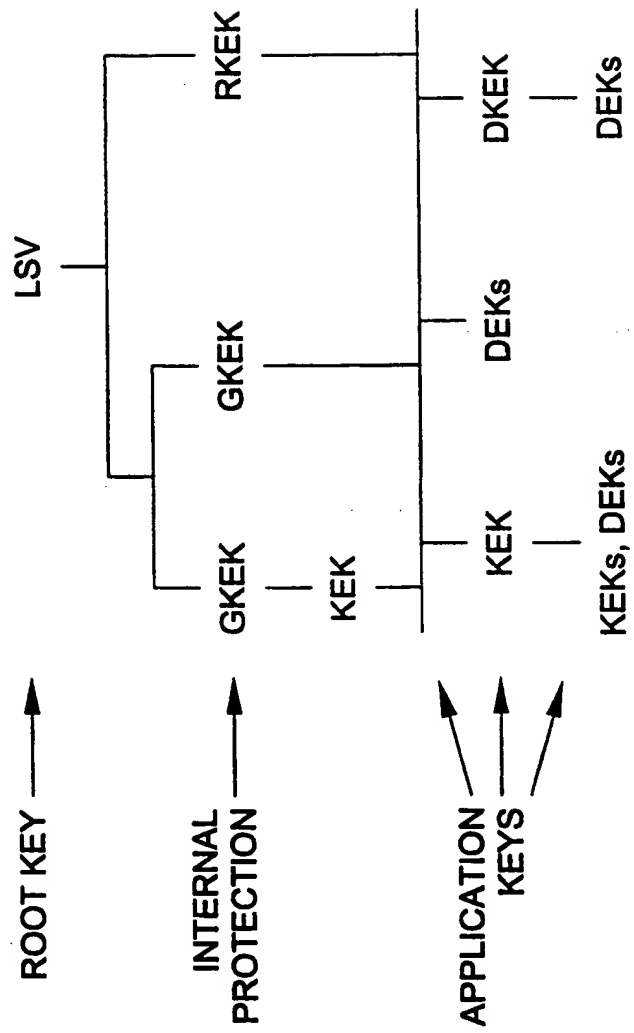


FIG-16 KEK HEIRARCHY

FIG-17 SYMMETRIC KEY WEAKENING ALGORITHM

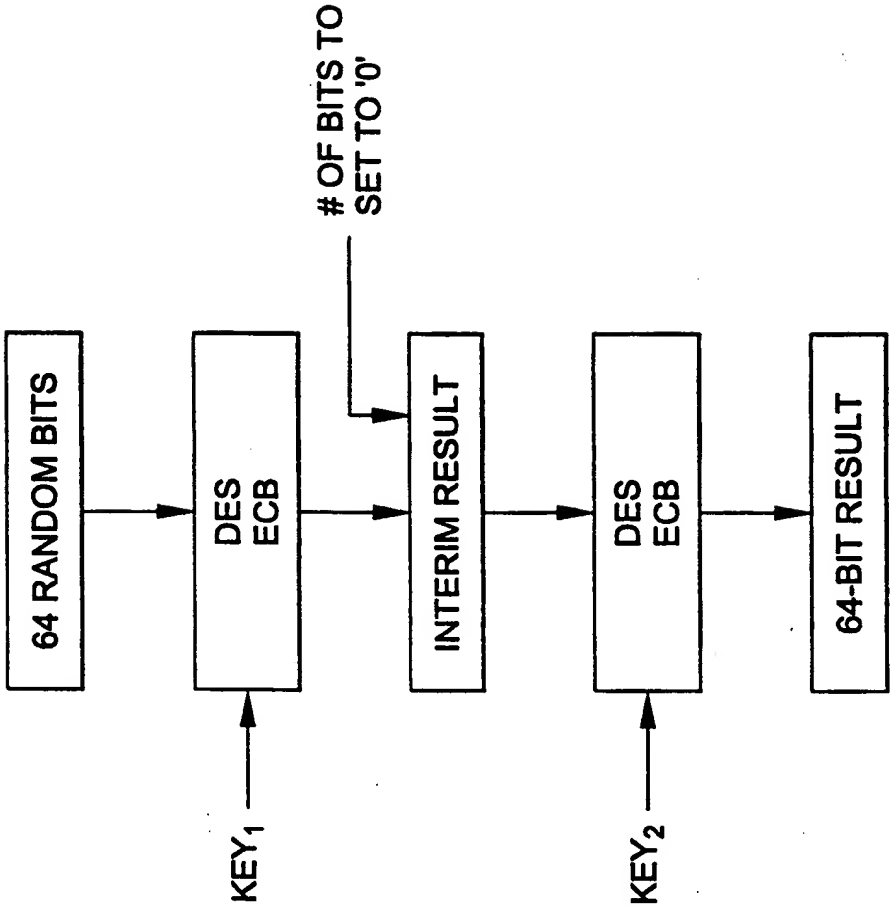


FIG-18 SYMMETRIC KEY HIERARCHY

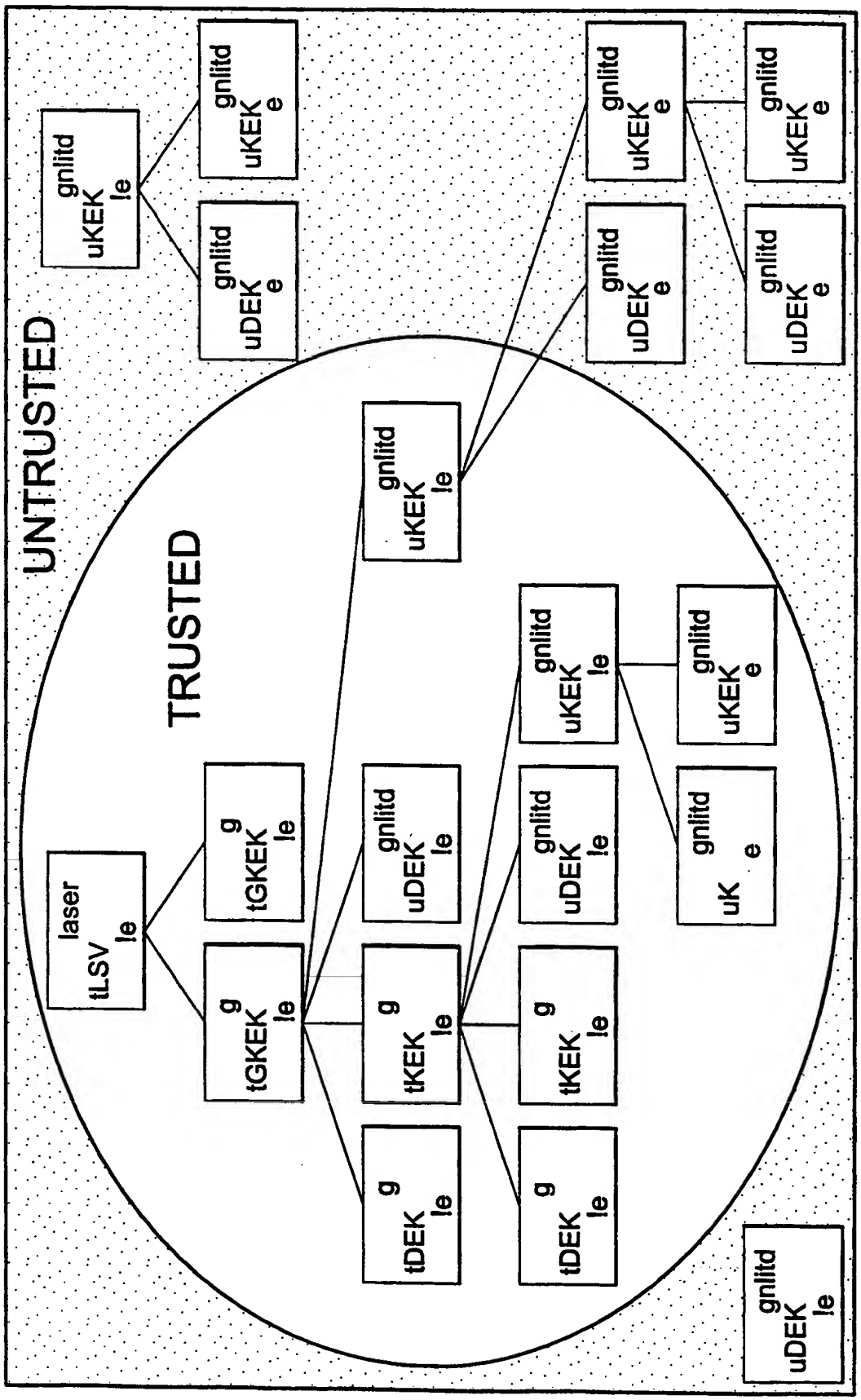


FIG-19 PCDB DATA TYPE DEFINITION

```
typedef struct _token_pcdb {
signblock token_signature;
UINT16    serial_number[CGX_SERIAL_NUMBER_LEN];
UINT16    length;
UINT16    reserved;
UINT16    pcdb[CGX_PCDB_LEN];
UINT16    reserved1;
} token_pcdb;
```

FIG-20 KERNEL BLOCK

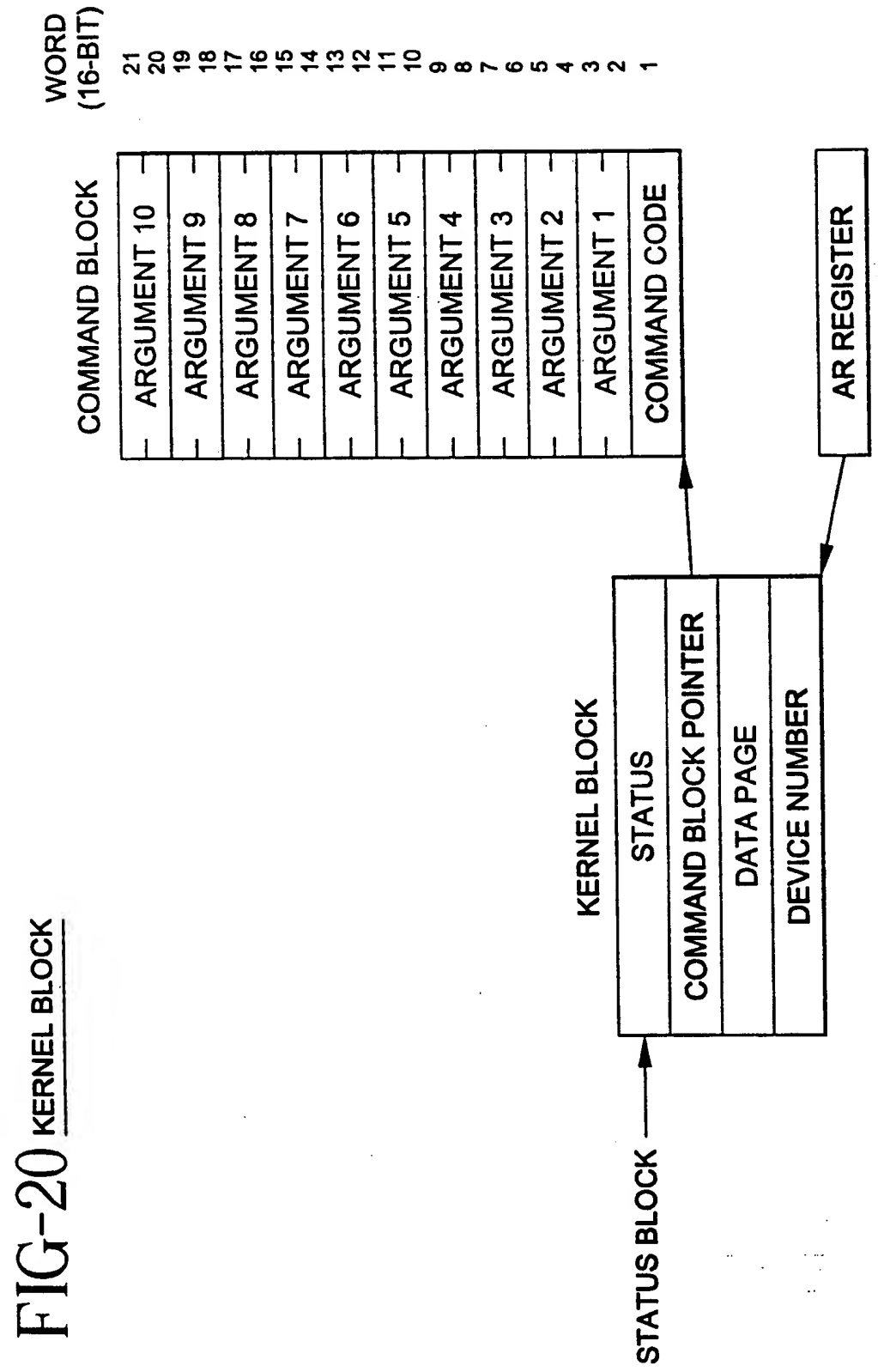


FIG-21 KERNEL BLOCK OBJECT DEFINITION

```
typedef unsigned short UINT16; /* unsigned 16 bit integer */

typedef  struct _kernelblock {
    UINT16       DeviceNo;   /* command class and memory model */
    UINT16       dp;         /* ADI-2183 specific, data page of cmd/stat *
    cmdblock    *cb;         /* command block object, low memory first */
    UINT16       status;     /* command execution status */
} kernelblock;
```

FIG-22 COMMAND BLOCK OBJECT DEFINITION

```
typedef  (void *)  VPTR;     /* a pointer to any data type or object */

typedef  struct _cmdblock {
    unsigned short   cmd;                        /* micro command */
    VPTR             argument[CGX_MAX_ARGS]; /* argument list */
} cmdblock;
```

FIG-23 SECRET KEY OBJECT DEFINITION

```
typedef WORD16 unsigned short;              /* 16 bits of data */

typedef struct _secretkey {
       UINT16       extra;       /* for application's use */
       UINT16       type;        /* the secret key type */
       UINT16       length;      /* key length */
       UINT16       k[CGX_RAW_SECRET_KEY_HASH_LENGTH];/* secret key, salt, */;
                                                 /* plus some attributes & len */
                                                 /* plus a SHA-1 HASH digest */

} secretkey;
```


FIG-24 PUBLIC KEYSET OBJECT DEFINITION

```
typedef  void             *VPTR;
typedef  unsigned short WORD16;

typedef struct _publickey {
      WORD16          extra;      */ 16 bits of space reserved for the application */
      WORD16          type;       */ public key algorithm; defines keyset object types */
      WORD16          length;     */ modulus length of public key (in bits) */
      VPTR            modulus;    */ modulus object */
      VPTR            pubkey;     */ public key object */
      VPTR            privkey;    */ private key object */
} publickey;
```

FIG-25 DIFFIE-HELLMAN PUBLIC KEYSET OBJECTS

```
typedef  unsigned short  WORD16;

#define CGX_2048_BITS 128 /* To allocate 2048 bits          */
#define CGX_1024_BITS  64 /* To allocate 1024 bits          */
#define CGX_64_BITS     4 /* To allocate   64 bits (salt) */

typedef UINT16 Twokbits[CGX_2048_BITS];
typedef UINT16 Onekbits[CGX_1024_BITS];
typedef UNIT16 Saltbits[CGX_64_BITS];

/* Diffie-Hellman Public Keyset Objects */
typedef struct _DHmodulus {
               WORD16 n[128];     /* max 2048-bit modulus                */
               WORD16 g[128];     /* max 2048-bit base                   */
} DHmodulus;

typedef struct _DHpubkey {
               WORD16 Y[128];     /* max 2048-bit public key             */
} DHpubkey;

typedef struct _DHprivkey {
               WORD16 salt[4];    /* salt for covering purposes          */
               WORD16 y[128];     /* max 2048-bit private key            */
} DHprivkey;
```

FIG-26 RSA PUBLIC KEYSET OBJECTS

```
typedef  unsigned short  WORD16;

typedef struct _RSAmodulus {
               WORD16    n[128];    /* maximum 2048 bit modulus          */
} RSAmodulus;

typedef struct _RSApubkey {
               WORD16    e[2];      /* maximum 32 bit public key         */
} RSApubkey;

typedef struct _RSAprivkey {
               WORD16    salt[4]    /* rndm data for covering purposes   */
               WORD16    p[64];     /* 1024 bit max prime modulus        */
               WORD16    q[64];     /* 1024 bit max prime modulus        */
               WORD16    dP[64];    /* 1024 bit max, d mod (p-1)         */
               WORD16    dQ[64];    /* 1024 bit max, d mod (q-1)         */
               WORD16    qInv[64]; /* 1024 bit max, q^-1 mod p           */
               WORD16    d[128];    /* maximum 2048 bit private key      */
} RSAprivkey;
```

FIG-27 DSA PUBLIC KEYSET OBJECTS

```
typedef  unsigned short  WORD16;

typedef struct _DSAmodulus {
               WORD16    p[128];   /* maximum 2048 bit modulus */
               WORD16    q[10];    /* fixed 160 bit modulus */
               WORD16    g[128];   /* maximum 2048, the generator */
} DSAmodulus;

typedef struct _DSApubkey {
               WORD16    y[128];        /* maximum 2048 bits public key */
} DSApubkey;

typedef struct _DSAprivkey {
               WORD16    salt[2]   /* rndm data for covering purposes */
               WORD16    x[10];    /* fixed private key size, 160 bits */
} DSAprivkey;
```

FIG-28 DSA DIGITAL SIGNATURE OBJECT DEFINITION

```
typedef unsigned short WORD16;     /* 16 bits of data */
typedef void           *VPTR;      /* pointer to any object */

typedef  struct _signblock {
         WORD16 r[10]; /* r = (g^k mod p) mod q */
         WORD16 s[10]; /* s = (k^-1 (H(m) + x r)) mod q */
} signature_block;
```

FIG-29

```
typedef unsigned short WORD16;     /* 16 bits of data */

typedef  struct _seedkey {
         WORD32 counter;  /* 32 bit counter to verify prime number */
         WORD16 seed[10]; /* 160 bit random data to generate a prime p and q */
} seedkey;
```

FIG-30 KEY CACHE REGISTER DATA TYPE DEFINITION

```
typedef unsigned short kcr;    /* 16 bits only */
```


FIG-31

```
typedef unsigned short WORD16;
typedef unsigned short BOOL;

typedef struct _crypto_cntxt {
    unsigned short config;      /* use one of the constants defined in the document */
                                /* titled, Cryptic Command Interface Specification */
    kcr5            key;        /* KCR ID for secret key */
    WORD16          iv[4];      /* 64 bit for IV, feed-back register */
} crypto_cntxt;
```

FIG-32 ONE-WAY HASH CONTEXT STORE

```
typedef unsigned char BYTE;
typedef unsigned short WORD16;
typedef unsigned long WORD32;

typedef struct {
    WORD32 count[2];        /* number of bits, module 2^64 (lsb first) */
    WORD32 state[4];        /* state (ABCD)                            */
    UINT16 buffer[32];      /* input buffer                            */
} MD5_CTX;

typedef struct {
    WORD32 count[2];        /* 64-bit bit count                        */
    WORD32 digest[5];       /* Message digest                          */
    WORD32 data[16];        /* SHS data buffer                         */
} SHS_INFO;

typedef struct _hash_cntxt {
    WORD16 algorithm;       /* Either CGX_MD5_A or CGX_SHS_A */
    BYTE   *digest;         /* Pointer to message digest [either
                             * 128-bits for MD5 or 160-bits for SHS.
                             */

    /* The following buffer stores the algorithm's state
     * information. (96 bytes)
     */
   union {
       MD5_CTXmd5;
       SHS_INFOshs;
     state
} hash_cntxt;
```

FIG-33 EXAMPLE OF CGX WRAP CODE AND COMMAND INTERFACE

```
#include "cgx.h"

#define cgx_encrypt(kb, datain_pg, datain, dataout_pg, dataout, size, cryptoblock) {\
                    kb->cb->cmd          =  CGX_ENCRYPT; \
                    kb->cb->agrument[0] = (VPTR)cryptoblock; \
                    kb->cb->agrument[1] = (VPTR)datain_pg; \
                    kb->cb->agrument[2] = (VPTR)datain; \
                    kb->cb->agrument[3] = (VPTR)size; \
                    kb->cb->agrument[4] = (VPTR)dataout_pg; \
                    kb->cb->agrument[5] = (VPTR)dataout; \
                    cgx_transfer_secure_kernel(kb); )

boolean
encrypt_packet_ecb(unsigned char *data, unsigned size, kcr key, kernelblock *kb)
{
     crypto_cntxt *cb;

     cb = (crypto_cntxt *)malloc(sizeof(crypto_cntxt));

     cb->config    = CGX_DES_ECB_M;
     cb->iv        = (WORD16 *)NULL;
     cb->key1      = key;

     cgx_encrypt(kb, 0, data, 0, data, size, cb); // request CGX Kernel to perform
                                                  // encryption of the data packet, copy
                                                  // of result is written back over src.

     free((unsigned char *)cb);
     /* check the result of the operation */
     if( kb->sb->status == CGX_SUCCESS )
            return TRUE;
     else
            return FALSE;
}
```

FIG-34 CGX OVERLAY TABLE DEFINITION

```
typedef unsigned short WORD16;              /* 16 bit data */
typedef void          (*CGX_FUNC) (void); /* function ptr */

typedef struct _cgx_overlay_tuple {
      CGX_FUNC cgxf;    /* CGX overlay operation */
      WORD16   control; /* control variable: preempt */
} cgx_overlay_tuple;
```

FIG-35 KCS OBJECT DEFINITION

```
#defineCGX_KCS_VERSION0x0001
#defineCGX_KCS_RESERVED_COUNT2
typedef struct _kcs {
UINT16 version;                                  /* version of KCS string */
UINT16  reserved[CGX_KCS_RESERVED_COUNT];/* Reserved fields */
UINT16  rng_alarm_count_threshhold;              /* RNG Alarm Count Threshhold */
UINT16  rng_enable;                              /* Enable RNG Clock */
UINT16  kernel_blocked;                          /* address of semaphore */
UINT16  kernel_dsp_dma;                          /* address of semaphore */
UINT16  kernel_dsp_dma_ifc;                      /* IFC for knl DMA release */
UINT16  dma_max_dwords;                          /* Max dwords knl DMA trans */
UINT16  kernel_dsp_hashenc;                      /* address of semaphore */
UINT16  kernel_dsp_hashenc_ifc;                  /* IFC for knl HASHENC rls */
UINT16  kernel_host_hashenc;                     /* enable bus grant/ack */
UINT16  kernel_host_hashenc_ifc:                 /* IFC for knl HASHENC rls */
UINT16  hashenc_iv;                              /* IV write disable cntl */
UINT16  hashenc_cntl;                            /* Cntl who owns the HASHENC */
UINT16  fips140_1;                               /* Enable fips140.1 mode */
UINT16  flipsha;                                 /* Control flips of SHA md */
UINT16  flipgxy;                                 /* Control flips g^xy of DH */
UINT16  rng_control;                             /* Used to control the RNG */
UINT16  rng_config;                              /* Used to configure the RNG */
} kcs;
```

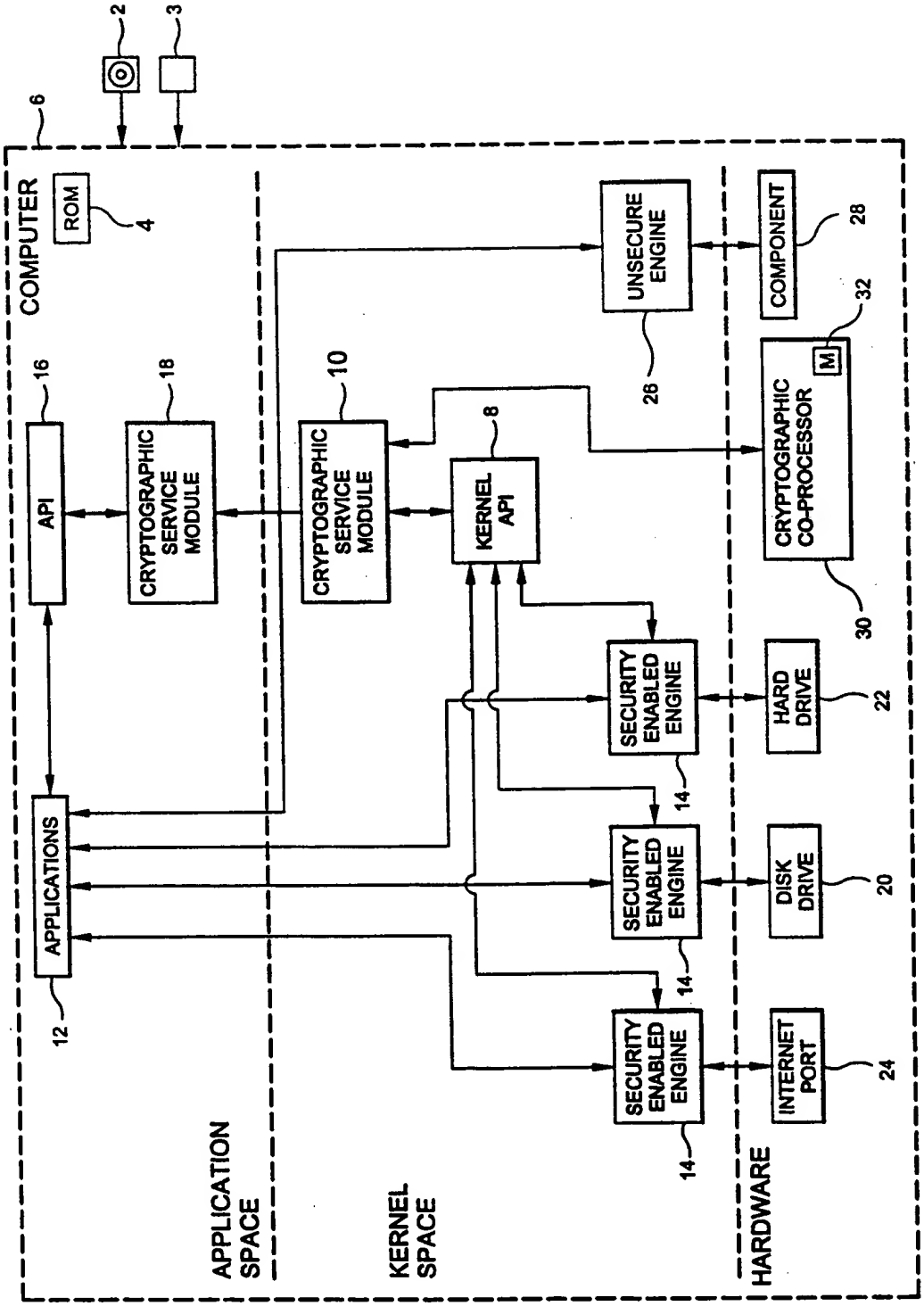

COMPUTER
6
2
3
ROM
4
API
16
CRYPTOGRAPHIC SERVICE MODULE
18
CRYPTOGRAPHIC SERVICE MODULE
10
KERNEL API
8
UNSECURE ENGINE
26
COMPONENT
28
CRYPTOGRAPHIC CO-PROCESSOR
M
32
30
SECURITY ENABLED ENGINE
14
HARD DRIVE
22
SECURITY ENABLED ENGINE
14
DISK DRIVE
20
SECURITY ENABLED ENGINE
14
INTERNET PORT
24
APPLICATIONS
12
APPLICATION SPACE
KERNEL SPACE
HARDWARE

FIG-36

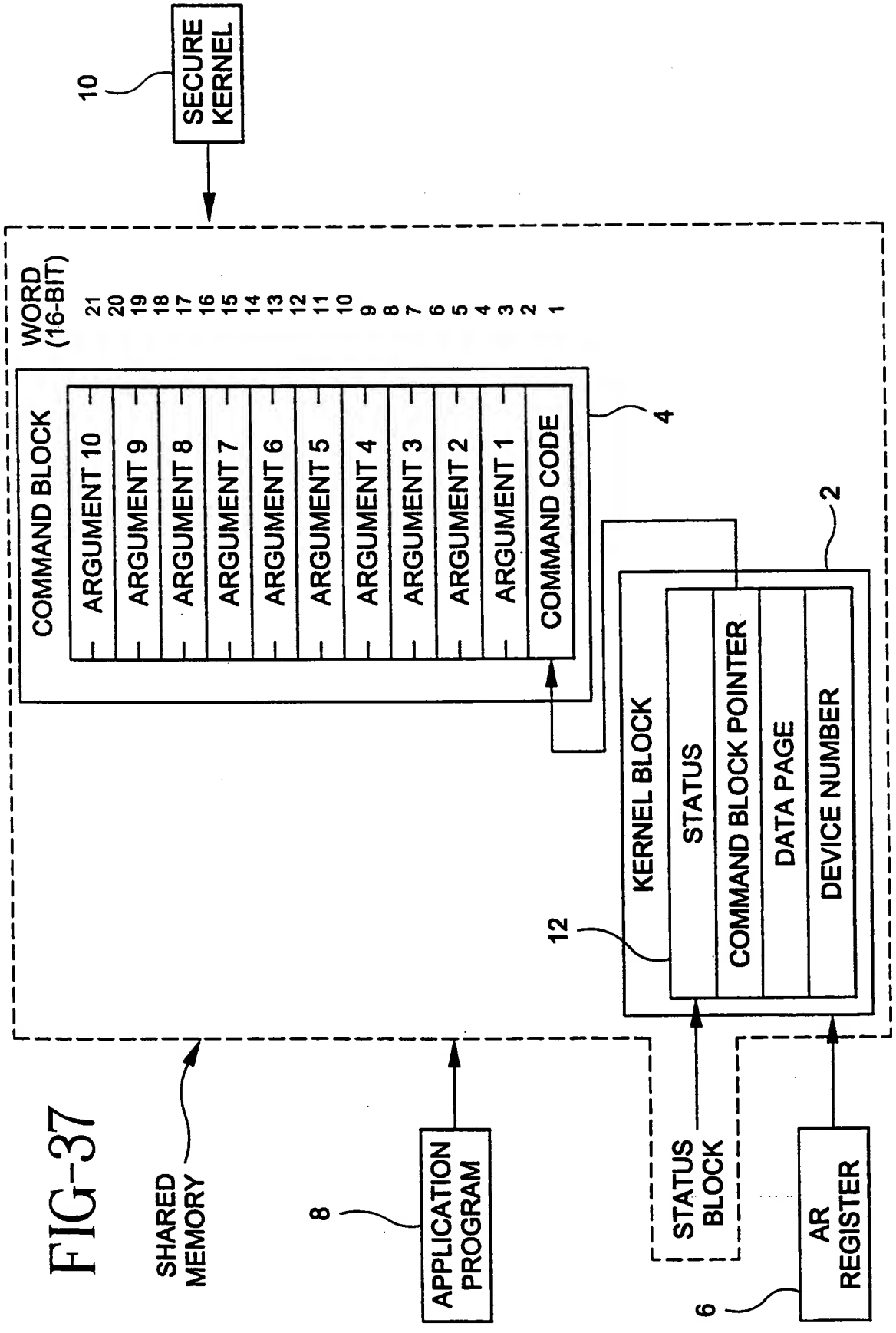
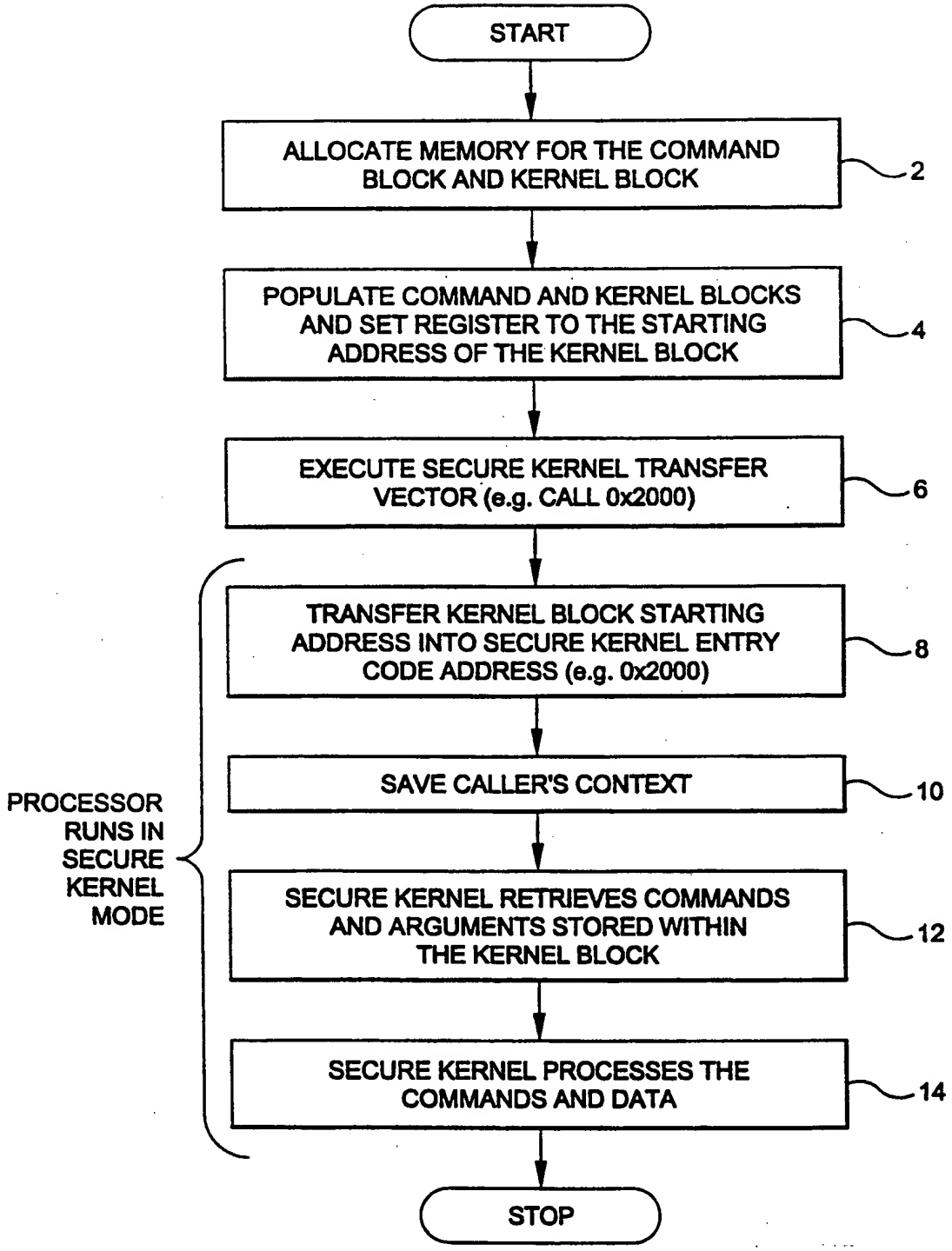

10
SECURE KERNEL
WORD (16-BIT)
21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
COMMAND BLOCK
ARGUMENT 10
ARGUMENT 9
ARGUMENT 8
ARGUMENT 7
ARGUMENT 6
ARGUMENT 5
ARGUMENT 4
ARGUMENT 3
ARGUMENT 2
ARGUMENT 1
COMMAND CODE
4
2
KERNEL BLOCK
STATUS
COMMAND BLOCK POINTER
DATA PAGE
DEVICE NUMBER
12
SHARED MEMORY
8
APPLICATION PROGRAM
STATUS BLOCK
6
AR REGISTER

FIG-37

FIG-38


START
ALLOCATE MEMORY FOR THE COMMAND BLOCK AND KERNEL BLOCK
2
POPULATE COMMAND AND KERNEL BLOCKS AND SET REGISTER TO THE STARTING ADDRESS OF THE KERNEL BLOCK
4
EXECUTE SECURE KERNEL TRANSFER VECTOR (e.g. CALL 0x2000)
6
PROCESSOR RUNS IN SECURE KERNEL MODE
TRANSFER KERNEL BLOCK STARTING ADDRESS INTO SECURE KERNEL ENTRY CODE ADDRESS (e.g. 0x2000)
8
SAVE CALLER'S CONTEXT
10
SECURE KERNEL RETRIEVES COMMANDS AND ARGUMENTS STORED WITHIN THE KERNEL BLOCK
12
SECURE KERNEL PROCESSES THE COMMANDS AND DATA
14
STOP

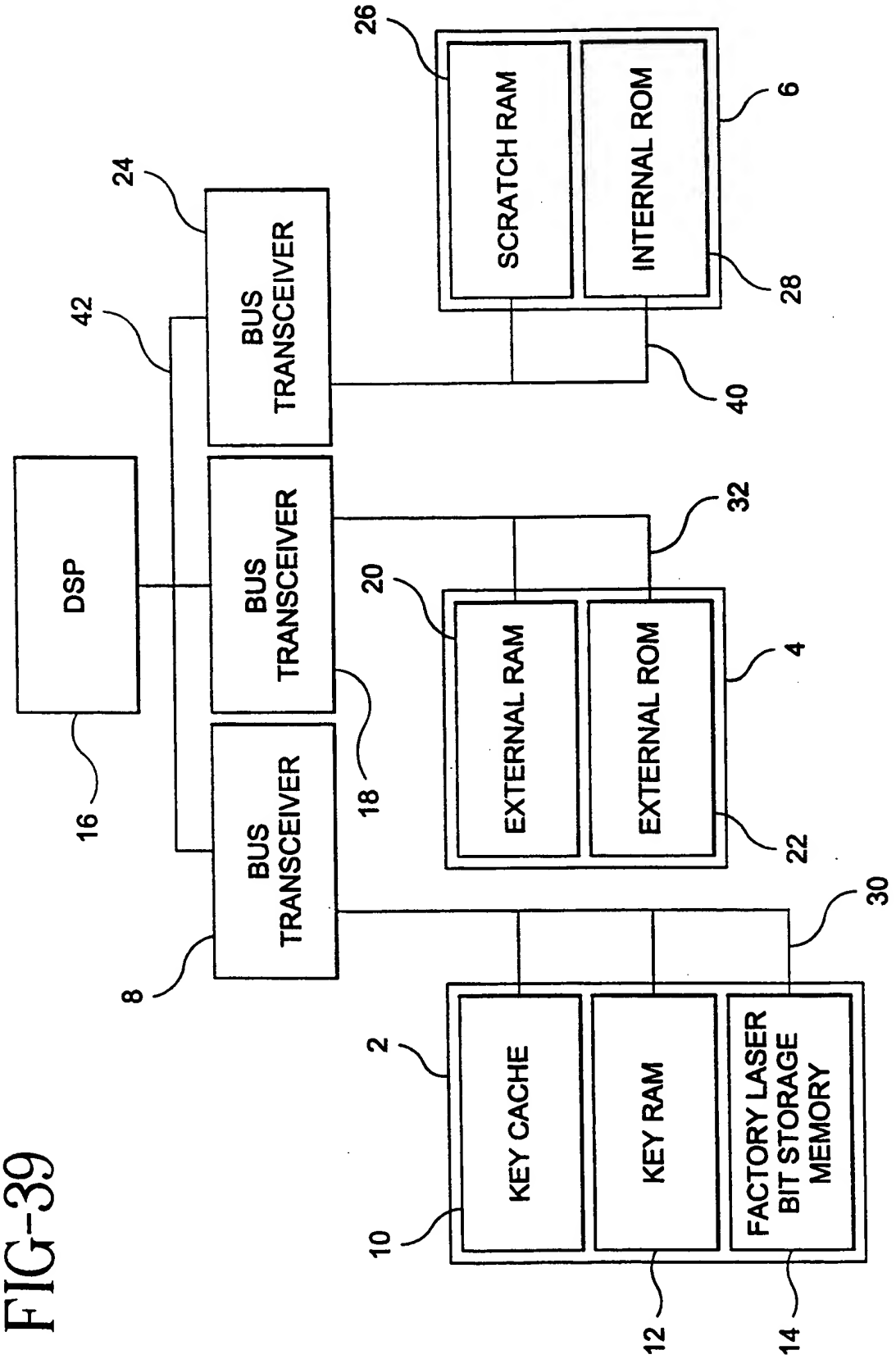

26
SCRATCH RAM
INTERNAL ROM
6
28
24
BUS TRANSCEIVER
42
40
DSP
BUS TRANSCEIVER
32
20
EXTERNAL RAM
EXTERNAL ROM
4
22
16
18
BUS TRANSCEIVER
8
30
2
KEY CACHE
KEY RAM
FACTORY LASER BIT STORAGE MEMORY
10
12
14

FIG-39

FIG-40

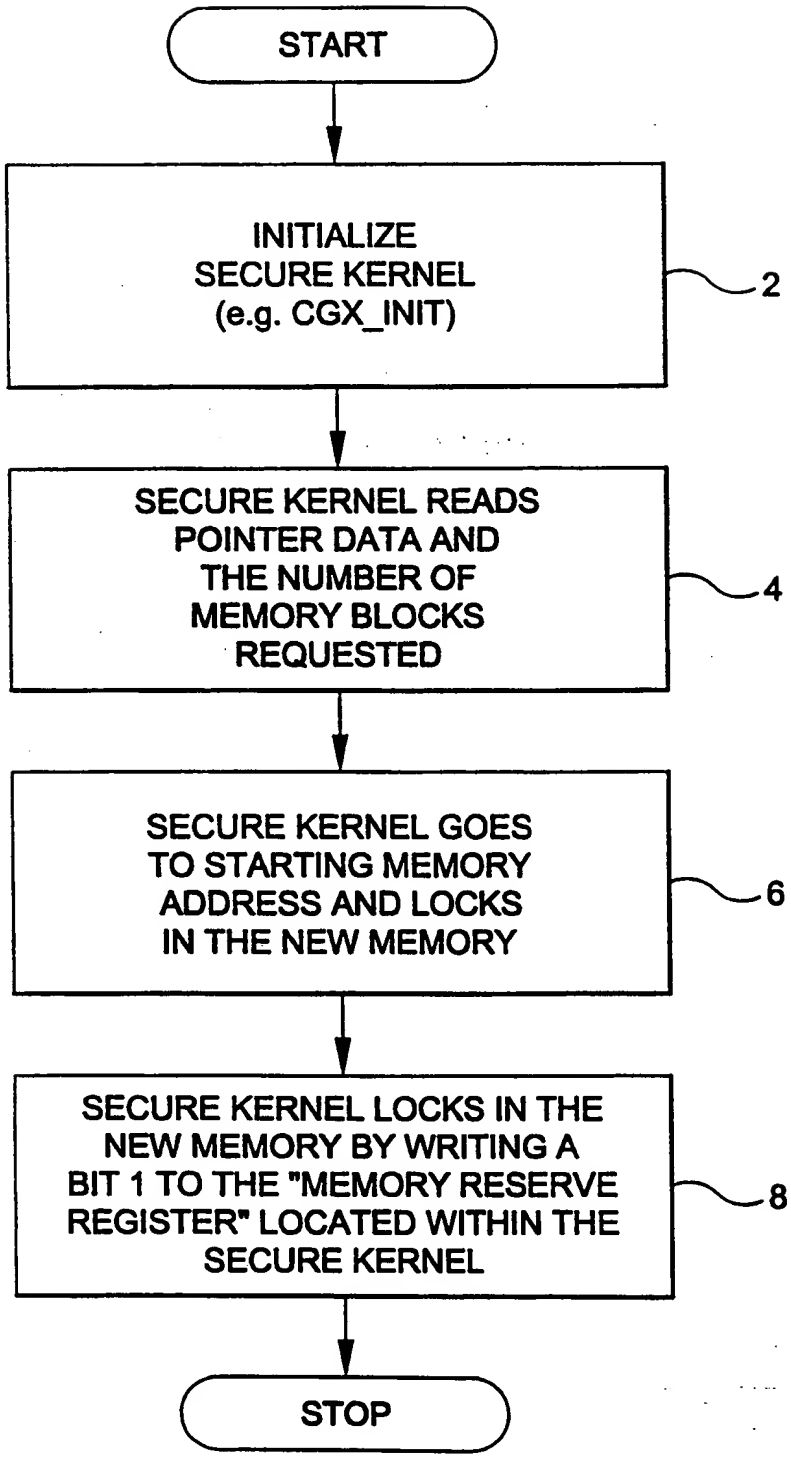

START
INITIALIZE SECURE KERNEL (e.g. CGX_INIT)
2
SECURE KERNEL READS POINTER DATA AND THE NUMBER OF MEMORY BLOCKS REQUESTED
4
SECURE KERNEL GOES TO STARTING MEMORY ADDRESS AND LOCKS IN THE NEW MEMORY
6
SECURE KERNEL LOCKS IN THE NEW MEMORY BY WRITING A BIT 1 TO THE "MEMORY RESERVE REGISTER" LOCATED WITHIN THE SECURE KERNEL
8
STOP

FIG-41 MEMORY LOCKING WITH xM RESERVE REGISTERS

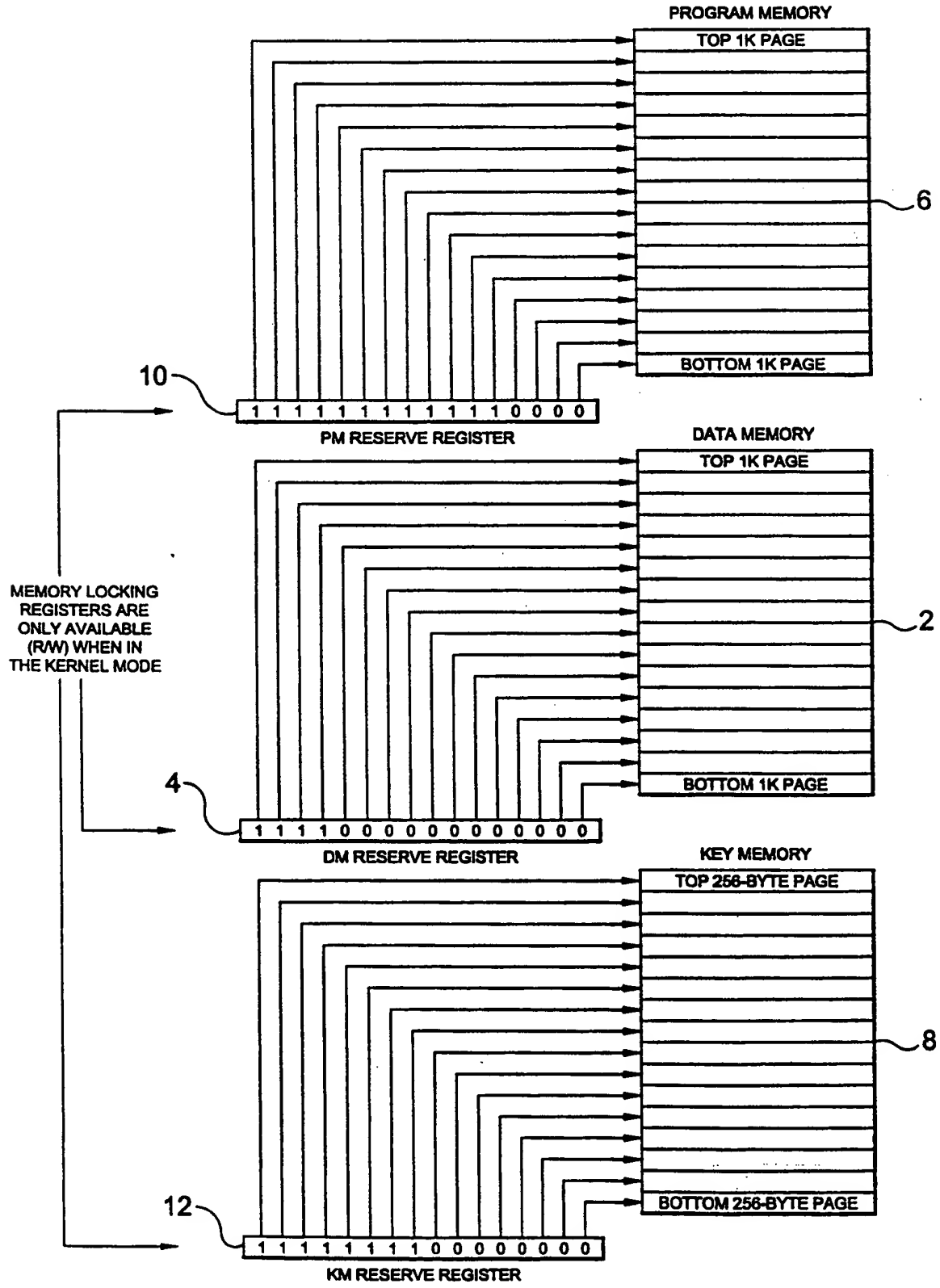


FIG-42

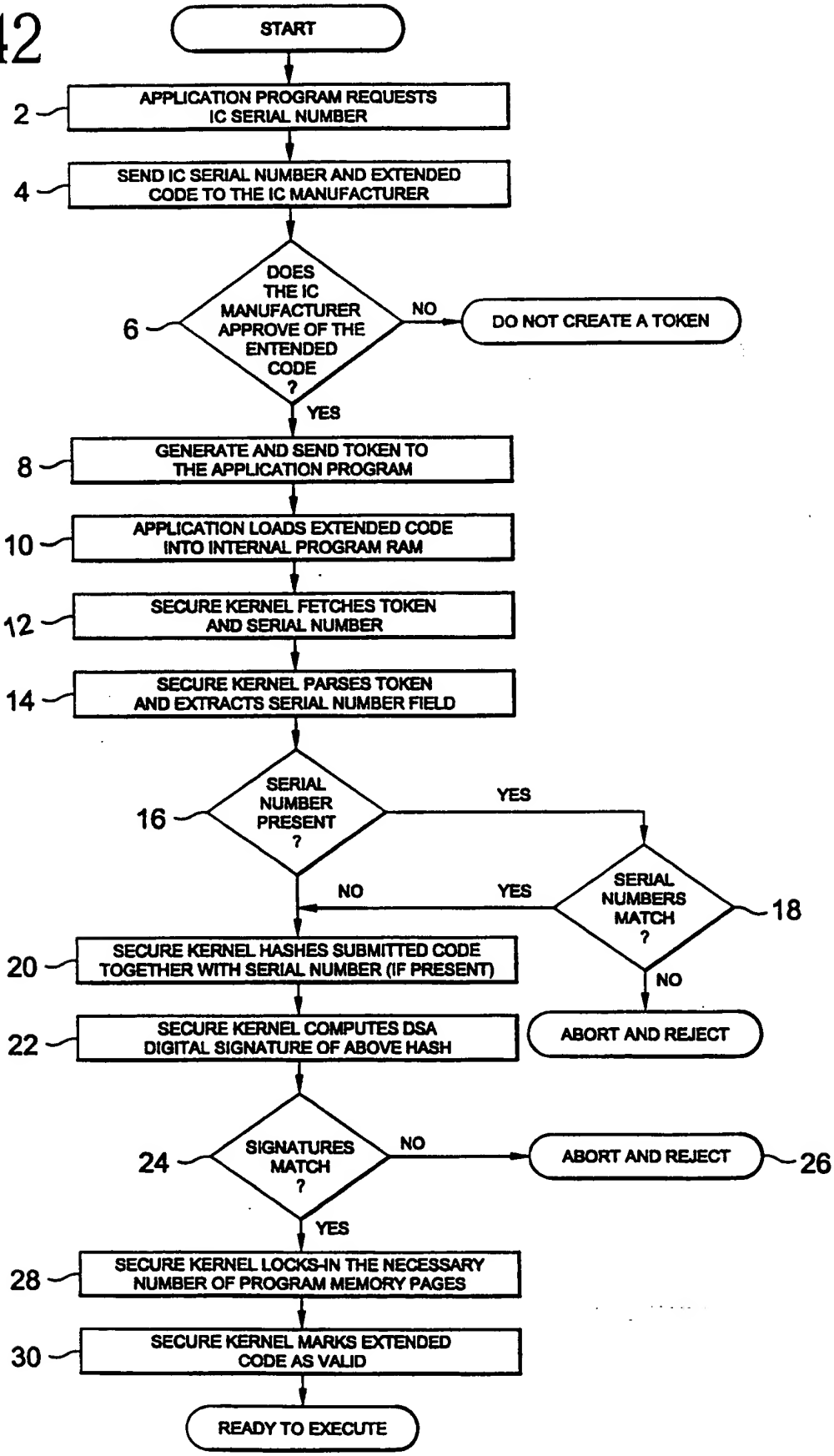

START
APPLICATION PROGRAM REQUESTS IC SERIAL NUMBER
2
SEND IC SERIAL NUMBER AND EXTENDED CODE TO THE IC MANUFACTURER
4
DOES THE IC MANUFACTURER APPROVE OF THE ENTENDED CODE ?
6
NO
DO NOT CREATE A TOKEN
YES
GENERATE AND SEND TOKEN TO THE APPLICATION PROGRAM
8
APPLICATION LOADS EXTENDED CODE INTO INTERNAL PROGRAM RAM
10
SECURE KERNEL FETCHES TOKEN AND SERIAL NUMBER
12
SECURE KERNEL PARSES TOKEN AND EXTRACTS SERIAL NUMBER FIELD
14
SERIAL NUMBER PRESENT ?
16
YES
SERIAL NUMBERS MATCH ?
18
NO
YES
NO
ABORT AND REJECT
SECURE KERNEL HASHES SUBMITTED CODE TOGETHER WITH SERIAL NUMBER (IF PRESENT)
20
SECURE KERNEL COMPUTES DSA DIGITAL SIGNATURE OF ABOVE HASH
22
SIGNATURES MATCH ?
24
NO
ABORT AND REJECT
26
YES
SECURE KERNEL LOCKS-IN THE NECESSARY NUMBER OF PROGRAM MEMORY PAGES
28
SECURE KERNEL MARKS EXTENDED CODE AS VALID
30
READY TO EXECUTE

FIG-43

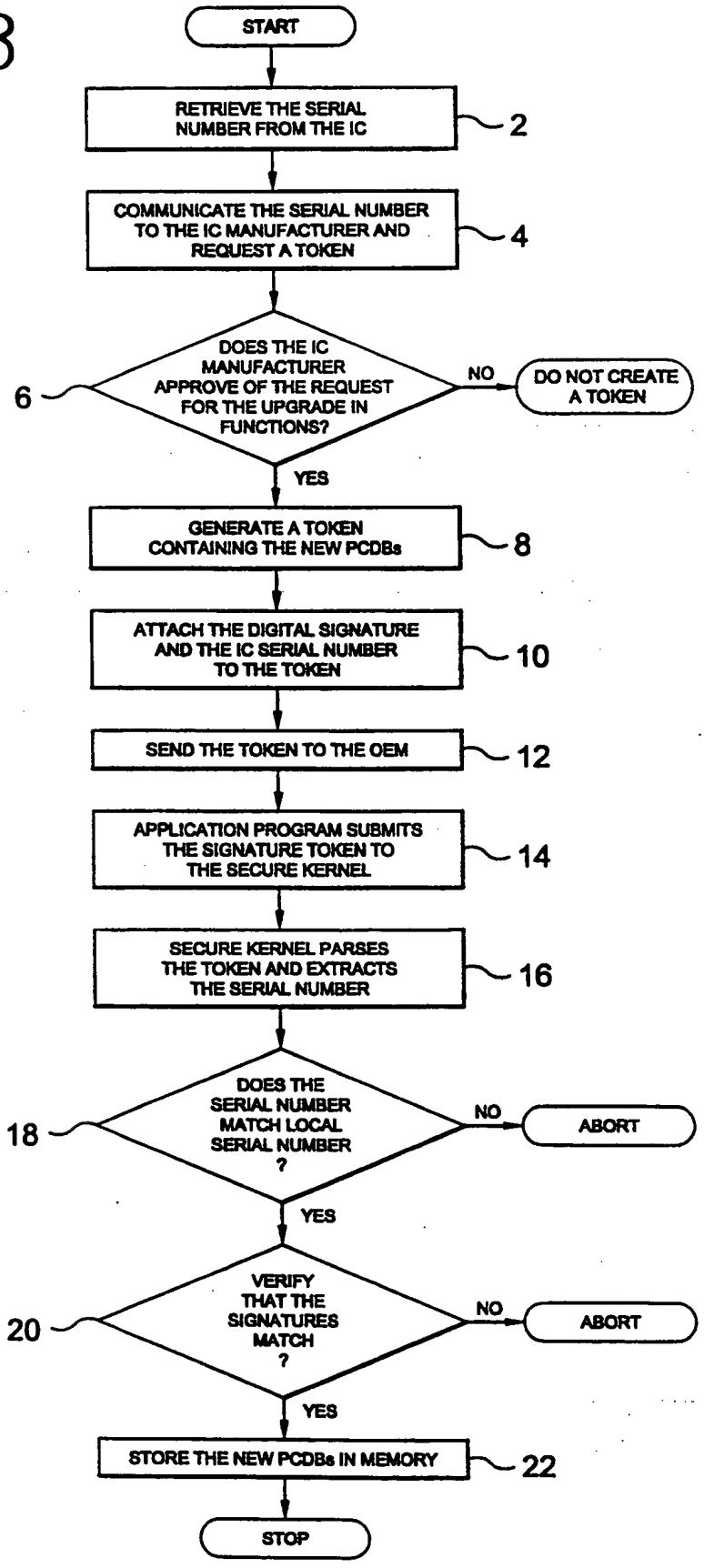
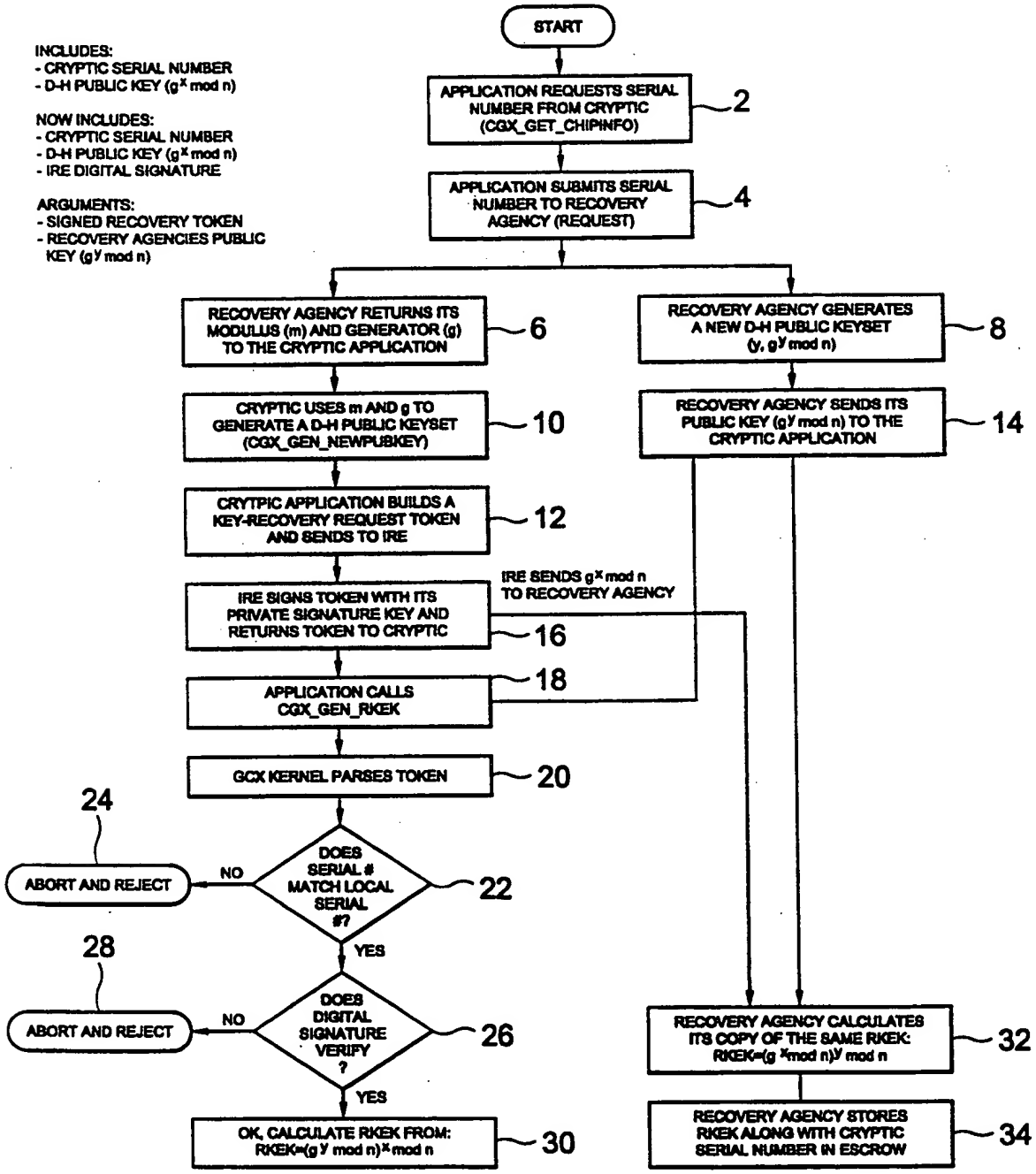

START
RETRIEVE THE SERIAL NUMBER FROM THE IC
2
COMMUNICATE THE SERIAL NUMBER TO THE IC MANUFACTURER AND REQUEST A TOKEN
4
DOES THE IC MANUFACTURER APPROVE OF THE REQUEST FOR THE UPGRADE IN FUNCTIONS?
6
NO
DO NOT CREATE A TOKEN
YES
GENERATE A TOKEN CONTAINING THE NEW PCDBs
8
ATTACH THE DIGITAL SIGNATURE AND THE IC SERIAL NUMBER TO THE TOKEN
10
SEND THE TOKEN TO THE OEM
12
APPLICATION PROGRAM SUBMITS THE SIGNATURE TOKEN TO THE SECURE KERNEL
14
SECURE KERNEL PARSES THE TOKEN AND EXTRACTS THE SERIAL NUMBER
16
DOES THE SERIAL NUMBER MATCH LOCAL SERIAL NUMBER ?
18
NO
ABORT
YES
VERIFY THAT THE SIGNATURES MATCH ?
20
NO
ABORT
YES
STORE THE NEW PCDBs IN MEMORY
22
STOP

FIG-44 KEY RECOVERY FLOW DIAGRAM



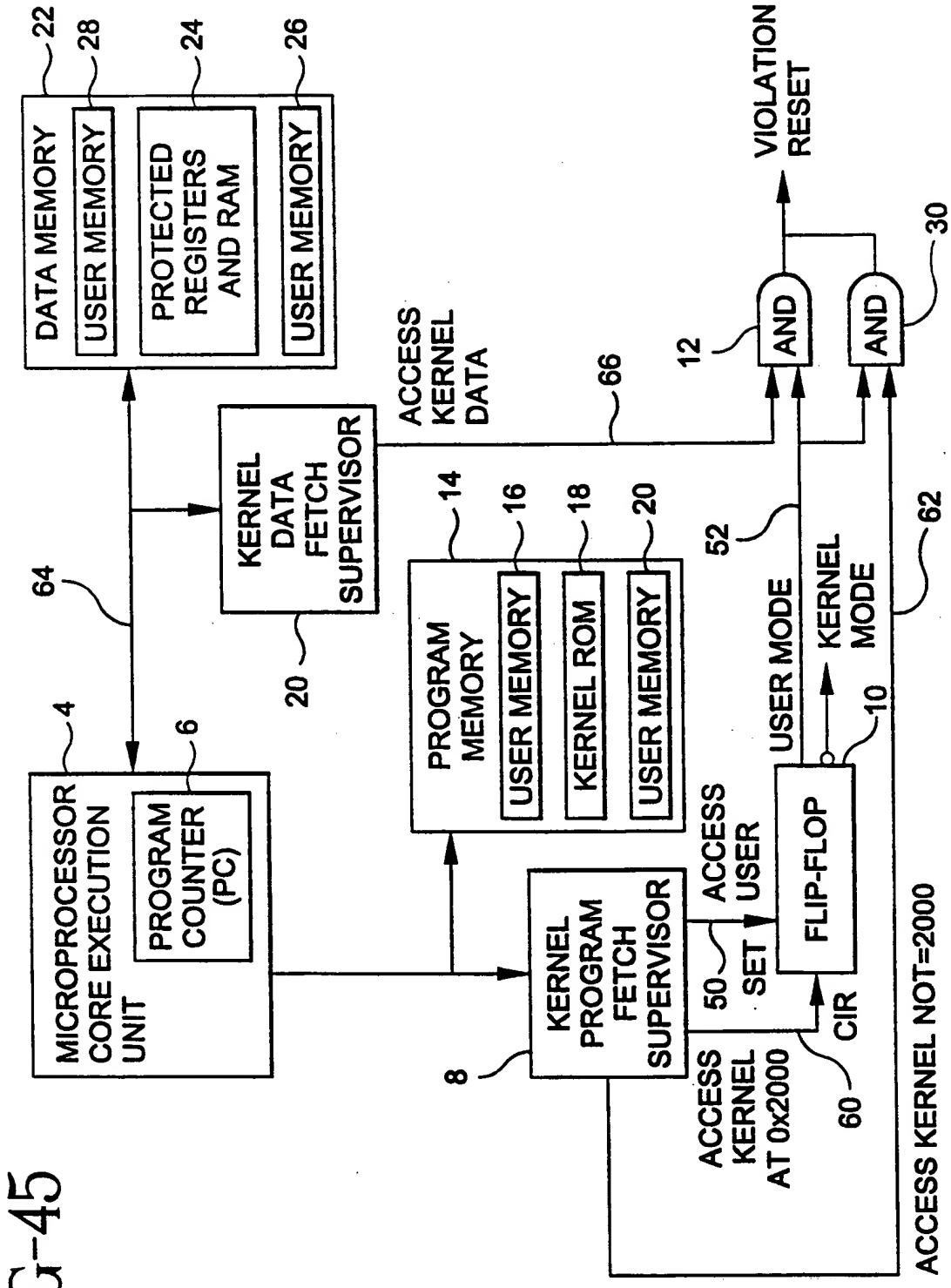

DATA MEMORY
22
USER MEMORY
28
PROTECTED REGISTERS AND RAM
24
USER MEMORY
26
VIOLATION RESET
AND
12
AND
30
ACCESS KERNEL DATA
66
KERNEL DATA FETCH SUPERVISOR
20
64
PROGRAM MEMORY
14
USER MEMORY
16
KERNEL ROM
18
USER MEMORY
20
52
62
MICROPROCESSOR CORE EXECUTION UNIT
4
PROGRAM COUNTER (PC)
6
USER MODE
KERNEL MODE
10
FLIP-FLOP
ACCESS USER
SET
50
KERNEL PROGRAM FETCH SUPERVISOR
8
ACCESS KERNEL AT 0x2000
CIR
60
ACCESS KERNEL NOT=2000

FIG-45

FIG-46

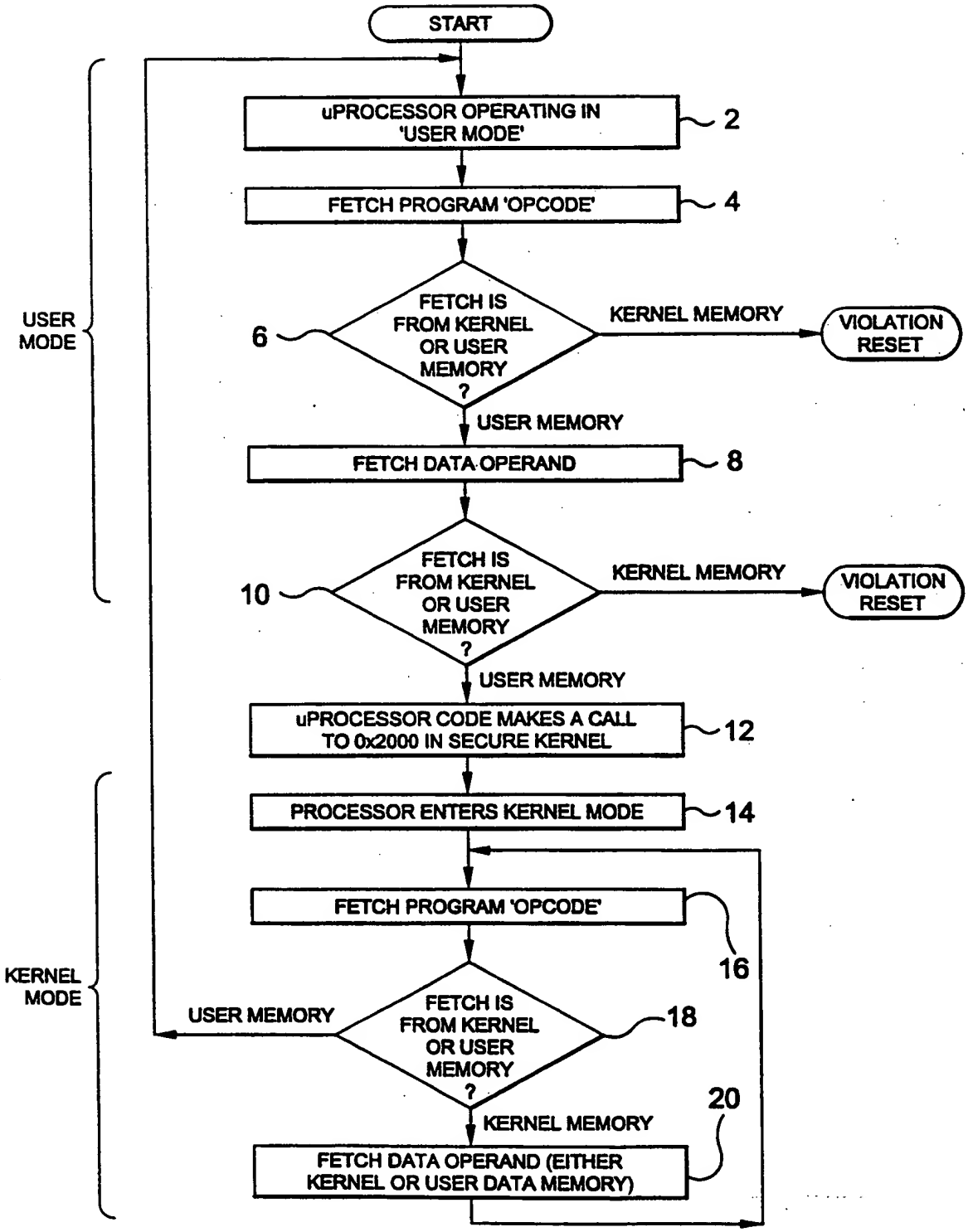

START
uPROCESSOR OPERATING IN 'USER MODE'
2
FETCH PROGRAM 'OPCODE'
4
USER MODE
FETCH IS FROM KERNEL OR USER MEMORY ?
6
KERNEL MEMORY
VIOLATION RESET
USER MEMORY
FETCH DATA OPERAND
8
FETCH IS FROM KERNEL OR USER MEMORY ?
10
KERNEL MEMORY
VIOLATION RESET
USER MEMORY
uPROCESSOR CODE MAKES A CALL TO 0x2000 IN SECURE KERNEL
12
PROCESSOR ENTERS KERNEL MODE
14
FETCH PROGRAM 'OPCODE'
16
KERNEL MODE
USER MEMORY
FETCH IS FROM KERNEL OR USER MEMORY ?
18
20
KERNEL MEMORY
FETCH DATA OPERAND (EITHER KERNEL OR USER DATA MEMORY)

FIG-47

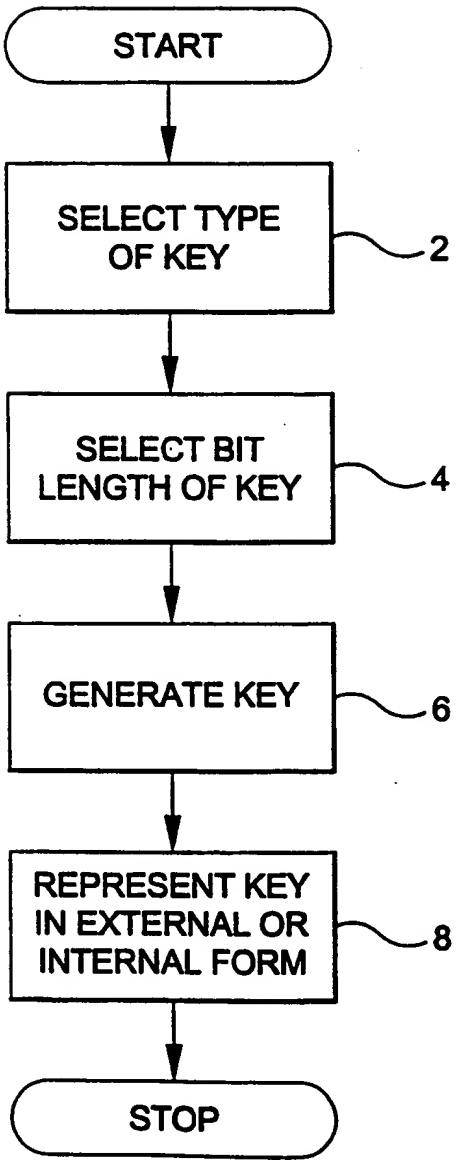
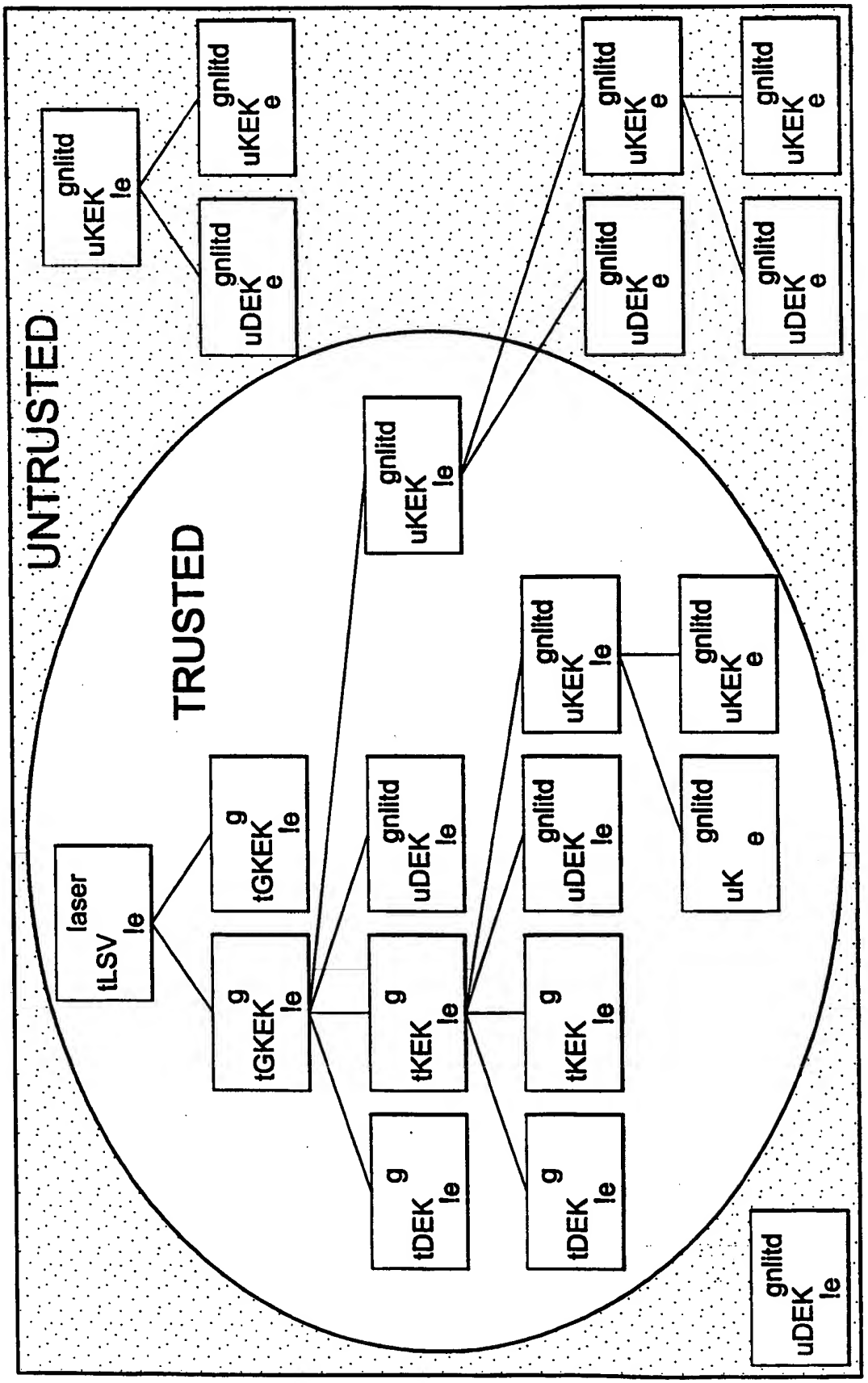

START
SELECT TYPE OF KEY
2
SELECT BIT LENGTH OF KEY
4
GENERATE KEY
6
REPRESENT KEY IN EXTERNAL OR INTERNAL FORM
8
STOP

FIG-48


UNTRUSTED
TRUSTED
laser tLSV le
g tGKEK le
g tGKEK le
g tKEK le
g tKEK le
g tDEK le
g tDEK le
gnlitd uDEK le
gnlitd uDEK le
gnlitd uDEK le
gnlitd uDEK le
gnlitd uKEK le
gnlitd uKEK le
gnlitd uKEK le
gnlitd uKEK le
gnlitd uKEK le
gnlitd uKEK le
gnlitd uK e
gnlitd uDEK le
gnlitd uDEK e
gnlitd uDEK e
gnlitd uDEK e
gnlitd uDEK e
gnlitd uKEK e
gnlitd uKEK e
gnlitd uKEK e
gnlitd uKEK e

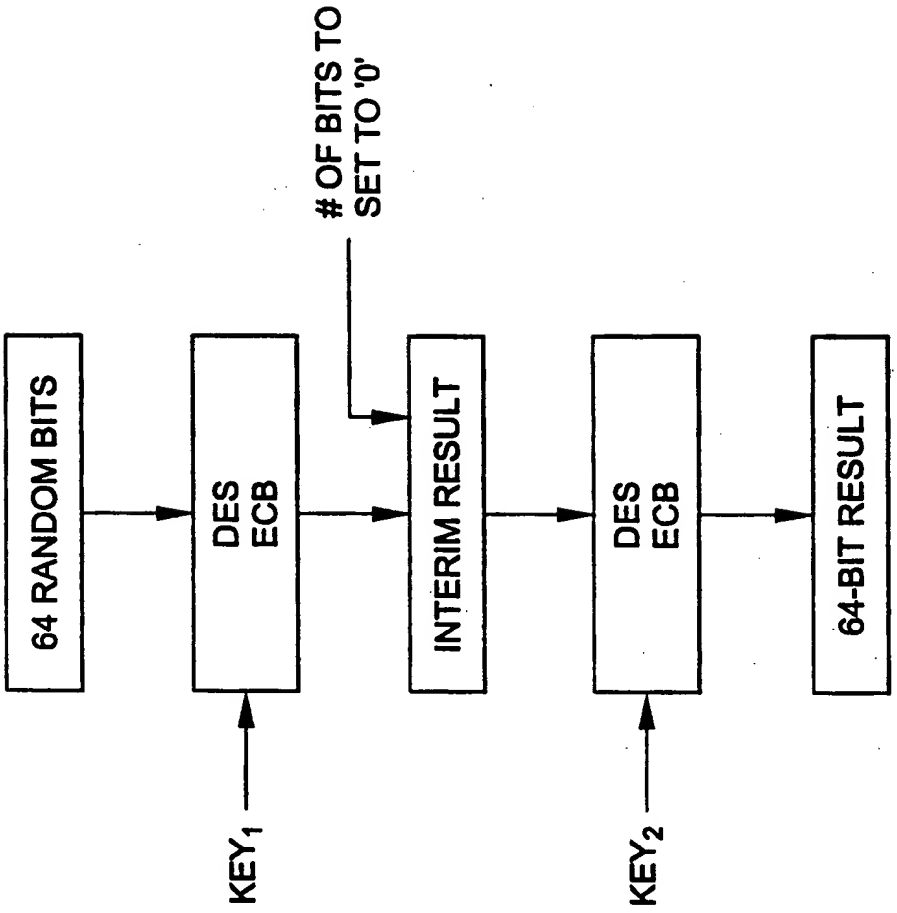

64 RANDOM BITS
DES ECB
OF BITS TO SET TO '0'
INTERIM RESULT
DES ECB
64-BIT RESULT
KEY1
KEY2

FIG-49

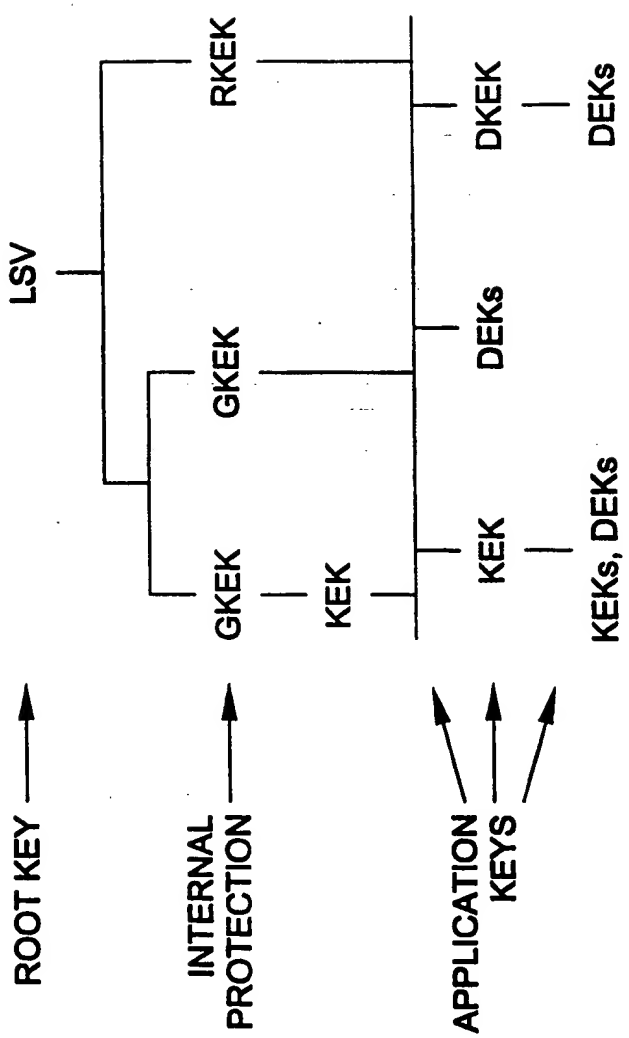

ROOT KEY
INTERNAL PROTECTION
APPLICATION KEYS
LSV
GKEK
KEK
RKEK
GKEK
KEK
DKEK
DEKs
KEK
KEKs, DEKs
DEKs

FIG-50